KATHOLIEKE UNIVERSITEIT
LEUVEN

**Arenberg Doctoral School of Science, Engineering & Technology**
Faculty of Engineering
Department of Computer Science

# Graph-based data mining for biological applications

**Leander Schietgat**

Promotors:
Prof. Dr. ir. H. Blockeel
Prof. Dr. ir. M. Bruynooghe

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

May 2010

# Graph-based data mining for biological applications

**Leander Schietgat**

Jury:
Prof. Dr. ir. H. Neuckermans, president
Prof. Dr. ir. H. Blockeel, promotor
Prof. Dr. ir. M. Bruynooghe, promotor
Prof. Dr. B. Demoen
Prof. Dr. ir. Y. Moreau
Dr. ir. J. Ramon
Prof. Dr. S. Džeroski
(Jožef Stefan Institute, Slovenia)
Prof. Dr. R. King
(Aberystwyth University, U.K.)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering

U.D.C. 681.3*I26

May 2010

© Katholieke Universiteit Leuven – Faculty of Engineering
Celestijnenlaan 200A, B-3001 Leuven (Belgium)

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2010/7515/57
ISBN 978-94-6018-218-1

# Beknopte samenvatting

Het onderzoek in deze thesis situeert zich in het domein van het relationeel leren. In het bijzonder stellen we leeralgoritmes voor die modellen bouwen voor gestructureerde gegevens op basis van grafen. Het belangrijkste doel van deze thesis is om de efficiëntie van relationele leeralgoritmes te verhogen, alsook hun toepasbaarheid op problemen uit de biologie en chemie.

In het eerste deel bestuderen we hiërarchische multi-label classificatie (HMC), een variant van classificatie waarbij een voorbeeld tot meerdere klassen kan behoren en waarbij de klassen georganiseerd zijn in een hiërarchie. Deze hiërarchie kan voorgesteld worden door een graaf en de uitvoer van een HMC-model bestaat uit één of meerdere paden van deze graaf. Een belangrijke toepassing van HMC is het voorspellen van functies van genen. Het is bekend dat een gen meerdere functies kan hebben, terwijl biologen deze functies hebben ingedeeld in hiërarchieën. In plaats van een methode te gebruiken dat voor iedere klasse een onafhankelijk model leert, stellen we een methode voor dat één model leert dat alle klassen ineens voorspelt. We tonen aan dat deze methode in de context van beslissingsbomen resulteert in modellen die niet alleen efficiënter geleerd worden, maar die ook beter presteren op het vlak van predictieve performantie, complexiteit en interpreteerbaarheid. Als we gaan vergelijken met state-of-the-art technieken voor het voorspellen van functies van genen stellen we vast dat de voorgestelde HMC-methode een hogere efficiëntie en een vergelijkbare predictieve performantie heeft.

In het tweede deel beschouwen we leeralgoritmes waarvan de invoer voorgesteld wordt door grafen. De toepassing die we hier voor ogen hebben is het leren van structuur-activiteitsrelaties (SAR). Het doel van SAR is om eigenschappen van moleculen te voorspellen aan de hand van hun chemische structuur. Om de leeralgoritmes efficiënter te maken, buiten we specifieke eigenschappen uit van moleculaire grafen. Doordat de meeste moleculen voorgesteld kunnen worden door outerplanaire grafen en omdat het blok-en-brug-behoudende (BBP) subgraaf isomorfisme een geschikte vergelijkingsoperator blijkt voor SAR, kunnen we een polynomiaal algoritme ontwikkelen dat een maximaal gemeenschappelijke subgraaf van twee outerplanaire grafen berekent. We gebruiken dit algoritme om een metriek voor moleculen te bouwen en om patronen voor moleculen te genereren. Deze methodes blijken niet alleen efficiënter te zijn dan bestaande methodes, maar behalen ook een state-of-the-art predictieve performantie op SAR-problemen.

# Abstract

The research presented in this thesis is situated in the field of relational learning. More specifically, we propose learning algorithms for structured data that are able to construct models for which either the input or the output data consist of graphs. The main goal of this thesis is to improve the efficiency of such learning algorithms and to apply them to real-life problems from biology and chemistry.

In the first part, we study the task of hierarchical multi-label classification (HMC), a variant of classification where an example may belong to multiple classes and where the classes are organised in a hierarchy. This hierarchy can be represented as a single graph, so that the output of an HMC model, that is, the classes that are predicted, consists of one or more paths in this graph. A key application of HMC is gene function prediction. It is known that a gene may have multiple functions, while biologists have organised these functions into hierarchies. Instead of following an approach that learns an independent model for each class, we propose an approach that learns a single model that predicts all the classes at once. We show that, at least in the context of decision trees, this results in models that are not only learned more efficiently, but that are also superior in terms of predictive performance, model size and interpretability. Moreover, the proposed HMC decision tree approach is better than the state-of-the-art tools for gene function prediction, producing models that are efficiently learnable on large datasets and that reach a competitive predictive performance, while being easier to use.

In the second part, we consider the task of graph mining, in which the input data of the learning algorithm are represented as graphs. The application we focus on is the learning of structure-activity relationships (SAR). Here, the goal is to predict properties of molecules based on their atom-bond structure. In order to make the learning algorithms more efficient, we exploit specific properties of molecular graphs. Motivated by the fact that the majority of molecules can be represented as outerplanar graphs and that the block-and-bridge-preserving (BBP) subgraph isomorphism is a suitable matching operator in the SAR context, we propose a polynomial algorithm that computes a maximum common subgraph of two outerplanar graphs. We use this algorithm to construct a metric for molecules and to generate features for them. It turns out that the proposed methods are not only more efficient than existing graph mining algorithms, but also obtain a state-of-the-art predictive performance on several SAR tasks.

# Acknowledgements

*If you can meet with Triumph and Disaster*
*And treat those two impostors just the same;*

– Rudyard Kipling

A very wise person said that the lessons come from the journey, not from the destination. In my opinion, this could not be more true for the journey that is pursuing a Ph.D. The text in front of you is only the end result of an unbelievable adventure that I have experienced over the last few years. Along the way I did learn a lot of lessons, and I want to thank the people who have supported me.

First and foremost, I am extremely grateful to my promotors Hendrik Blockeel and Maurice Bruynooghe for having given me the opportunity to start a Ph.D. in the machine learning group. During the first few months Hendrik taught me everything I needed to know to become a good scientist: he learned me to ask the right questions, to see the story behind the numbers and curves, and even more importantly, he learned me how to communicate this story to the outside world. Although his time has become more and more precious over the years, he was always prepared to offer me guidance for any given problem. I am particularly amazed by his ability to meticulously go over a piece of text and then transform it without any visible effort into something that is half the length and twice as convincing. Besides his exceptional motivational skills, he is also a great person to have non-research related discussions with. Maurice, the founding father of our group, keeps a close eye on every DTAI member. Although he was not always familiar with the details of my work, when necessary he could immediately understand what I was doing and provide me with insightful views and useful suggestions for improvements. I also want to thank my daily supervisor Jan Ramon. He guided me through my very first research steps as a master student, and during my Ph.D. I could always count on his encyclopædic knowledge, on his help in writing proposals and for showing me how to effectively handle master students.

Apart from Hendrik, Maurice and Jan, I also want to thank the other members of my jury for reading and evaluating my text: Bart Demoen, Yves Moreau, my external members Sašo Džeroski and Ross King, and the chairman Herman

Neuckermans. I especially thank Yves Moreau for giving me the chance to present my work to his group and Sašo Džeroski for the fruitful cooperations and for the many valuable discussions.

As is usually the case, this thesis is not the accomplishment of one person. Therefore, I would like to sincerely thank all my co-authors who have contributed to this text: Jan Struyf, Celine Vens, Jan Ramon, Dragi Kocev, Sašo Džeroski, Fabrizio Costa and Luc De Raedt. I am extremely grateful of having had the opportunity to work with such professional and enthusiastic people. I also thank the anonymous reviewers of our papers whose remarks have led to many improvements of our work.

I want to thank Celine for the excellent team work, not only in our research, but also in organising scientific events and group activities. Although she has been abroad for the last year, her valuable opinion was always only two mouse clicks away. I am eternally indebted to JanS for helping me with major and minor programming issues and for his optimism and many motivational speeches, usually accompanied by some chocolates. When I started my Ph.D., Daan was the senior student of our office to who I could look up to and turn to for any kind of help, and I guess I have never stopped doing that. In that respect, I also owe a lot to Joaquin, who started writing his thesis a few weeks before me, and was the perfect victim to bother with Ph.D. practicalities. A special word goes to Tias, who's infinite enthusiasm has motivated me and helped me through the final months.

I would like to say a huge thank you to all of my current and former office mates, in order of appearance: Daan, Fabián, Snezhana, Tias, Beau, Celine, Jan, Kurt, Jelle and Ole. Thanks for making my time spent at the office so enjoyable! Over the years I have witnessed the group grow spectacularly and it is impossible to mention all of my colleagues here. To all of you, thank you for contributing to the amicable atmosphere within our group, for the entertaining lunch breaks at Alma, for the occasional poker evening and for being pleasant travel companions. Special thanks to Tias, Eduardo, Joaquin, Anneleen, Maarten and Álvaro who regularly joined me in much-needed lunch-break activities such as running and playing tennis.

For protecting me from the K.U.Leuven bureaucracy and helping me out with computer related problems, I thank the administrative and technical staff of the department, in particular Karin Michiels and Denise Brams. I gratefully acknowledge the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT Vlaanderen), which provided me with financial support for the past four years.

I want to thank my friends for their endless support and for providing the necessary distractions from my research. Karl, Ine, Marieke and Inge: thanks a million for always being there for me no matter what. Evelien, Jonathan, Tommy and Maarten: thanks for the priceless moments we share every time we meet. To the rest of my fellow informaticians and mathematicians who started their

studies at the Kulak: I'm glad we are still in touch and keep supporting each other. Thanks to Sibren, Steven, Christine and Isabel, not only for joining me once in a while alongside or opposite the (tennis) net, but also for the great off-court conversations. A special word of thanks goes to my housemates. They have probably been the key witnesses of my triumphs and disasters of the last few months.

Last but not least, I would like to express my deepest gratitude to my mom and dad for their love and support and for giving every possible opportunity in life. I also thank my sister and my grandmother for their continuous encouragements and the rest of my family, who each showed me that you can achieve anything if you set your mind to it.

Bedankt allemaal !

Leander Schietgat
Leuven, May 2010

# Contents

**Beknopte samenvatting** **i**

**Abstract** **iii**

**Acknowledgements** **v**

**Contents** **ix**

**List of Figures** **xiii**

**List of Tables** **xv**

**List of Algorithms** **xvii**

## Introduction 1

**1 Outline** **3**
- 1.1 Context . . . 3
  - 1.1.1 Artificial intelligence . . . 3
  - 1.1.2 Machine learning . . . 4
  - 1.1.3 Knowledge discovery and data mining . . . 6
  - 1.1.4 Biological applications of learning and mining . . . 7
- 1.2 Motivation and contributions . . . 8
- 1.3 Structure of the text . . . 10
- 1.4 Bibliographical note . . . 11

**2 Background** **13**
- 2.1 Introduction . . . 13
- 2.2 Structured data . . . 13
  - 2.2.1 Attribute-value data . . . 14
  - 2.2.2 Relational databases . . . 15

2.2.3 Graphs . . . 15
2.2.4 Logic programs . . . 20
2.3 Types of learning and mining . . . 22
2.3.1 Attribute-value learning . . . 22
2.3.2 Relational learning . . . 23
2.3.3 Propositionalisation . . . 25
2.4 Decision tree learning . . . 26
2.4.1 Decision trees . . . 26
2.4.2 Top-down induction of decision trees . . . 28
2.5 Measuring the quality of a learned model . . . 29
2.5.1 Accuracy . . . 31
2.5.2 ROC analysis . . . 31
2.5.3 Precision and recall . . . 33
2.6 Summary . . . 33

**I Structured Output Learning 35**

**Outline Part I 37**

**3 Decision Trees for Hierarchical Multi-label Classification 39**
3.1 Introduction . . . 39
3.2 Related work . . . 41
3.3 Decision tree approaches for HMC . . . 43
3.3.1 Formal task description . . . 43
3.3.2 Predictive clustering trees . . . 43
3.3.3 Clus-HMC: An HMC decision tree learner . . . 45
3.3.4 Clus-SC: Learning a separate tree for each class . . . 47
3.3.5 Clus-HSC: Learning a separate tree for each hierarchy edge 48
3.3.6 Comparison between Clus-HMC, Clus-SC and Clus-HSC 49
3.4 Hierarchies structured as DAGs . . . 50
3.4.1 Adaptations to Clus-HMC . . . 50
3.4.2 Adaptations to Clus-HSC . . . 51
3.5 Predictive performance measures for HMC . . . 52
3.5.1 Hierarchical loss . . . 52
3.5.2 Precision-recall based evaluation . . . 53
3.6 Experiments in yeast functional genomics . . . 55
3.6.1 Datasets . . . 56
3.6.2 Method . . . 58
3.6.3 Results . . . 59
3.7 Conclusions . . . 70

**4 Decision Tree Ensembles for Gene Function Prediction 73**
4.1 Introduction . . . 73
4.2 Related work . . . 75
4.2.1 Network based methods . . . 75
4.2.2 Kernel based methods . . . 76
4.2.3 Decision tree based methods . . . 77
4.3 Ensembles of HMC decision trees . . . 77
4.4 Experiments . . . 78
4.4.1 Datasets . . . 79
4.4.2 Method . . . 80
4.4.3 Results . . . 81
4.5 Conclusions . . . 90

**Conclusion Part I 95**

**II Structured Input Learning 97**

**Outline Part II 99**

**5 A Polynomial-time Maximum Common Subgraph Algorithm 101**
5.1 Introduction . . . 101
5.2 Graph theoretical concepts . . . 103
5.3 Computing a maximum common subgraph of two outerplanar graphs 107
5.3.1 Definitions and notations . . . 107
5.3.2 Generation of relevant subgraphs and computation of the parent-child relationships . . . 111
5.3.3 Bottom-up pairwise MCS computation of the relevant subgraphs . . . 115
5.3.4 Correctness proof sketch . . . 118
5.3.5 Time complexity . . . 122
5.4 Related work . . . 123
5.5 Experiments . . . 124
5.5.1 Dataset . . . 125
5.5.2 Method . . . 125
5.5.3 Results . . . 125
5.6 Conclusions . . . 126

**6 An Efficiently Computable Metric for Outerplanar Graphs 129**
6.1 Introduction . . . 129
6.2 Related work . . . 131
6.2.1 Descriptor-based methods . . . 131
6.2.2 Relational methods . . . 131

6.2.3 Propositionalisation methods . . . 132
6.3 A metric for outerplanar graphs . . . 133
6.3.1 Definition . . . 133
6.3.2 Using the maximum common subgraph notion to construct a metric . . . 133
6.4 Experiments . . . 134
6.4.1 Datasets . . . 135
6.4.2 Method . . . 136
6.4.3 Results . . . 137
6.5 Conclusions . . . 139

**7 Maximum Common Subgraph Sampling 143**
7.1 Introduction . . . 143
7.2 Using maximum common subgraphs as features . . . 144
7.2.1 Problem description . . . 145
7.2.2 Method . . . 145
7.2.3 Computational complexity . . . 146
7.3 Experiments . . . 147
7.3.1 Datasets . . . 148
7.3.2 State-of-the-art methods . . . 149
7.3.3 Method . . . 150
7.3.4 Results . . . 152
7.4 Discussion . . . 162
7.4.1 Drawbacks of the method . . . 163
7.4.2 Related work . . . 164
7.5 Conclusions . . . 165

**Conclusion Part II 167**

## Conclusions 169

**8 Thesis Summary and Further Work 171**
8.1 Thesis summary . . . 171
8.2 Further work . . . 174

**References 177**

**Index 190**

**Publication List 192**

**Biography 195**

# List of Figures


2.1 Two representations of the molecule aspirin . . . . . 17
2.2 Small part of the Gene Ontology . . . . . 18
2.3 Graph fragments occurring in molecules . . . . . 19
2.4 The hierarchy of graphs . . . . . 20
2.5 A hypothetical set of graph patterns . . . . . 26
2.6 Propositionalising graphs into vectors . . . . . 27
2.7 An example decision tree . . . . . 28
2.8 A binary decision tree . . . . . 29
2.9 Example ROC curve . . . . . 32

3.1 A small part of the hierarchical FunCat classification scheme . . . 40
3.2 A toy hierarchy . . . . . 46
3.3 Overview of the HMC, HSC and SC approach . . . . . 47
3.4 Adaptation of the CLUS-HSC approach towards DAG hierarchies . 52
3.5 Comparing weighting schemes for GO . . . . . 61
3.6 Examples of the AUPRC evaluation measures . . . . . 64
3.7 Average PR curves for FunCat and GO (hom) . . . . . 67
3.8 Average PR curves for FunCat and GO (seq) . . . . . 67
3.9 Example class-wise PR curves for FunCat and GO (hom) . . . . . 68
3.10 Difference in AUPRC versus class frequency for GO (hom) . . . . . 69

4.1 Example of a predictive clustering tree built on homology data . . 74
4.2 Comparison of AU($\overline{\text{PRC}}$) between CLUS-HMC and CLUS-HMC-ENS 82
4.3 Average PR curves for CLUS-HMC, CLUS-HMC-ENS and C4.5H/M 83
4.4 Comparison of $\overline{\text{AUPRC}}$ between CLUS-HMC and CLUS-HMC-ENS 84
4.5 Comparison of precision between C4.5H/M, CLUS-HMC and CLUS-HMC-ENS, at the recall obtained by C4.5H/M . . . . . 85
4.6 PR curve for class 29 on $D_4$ with FunCat annotations . . . . . 86
4.7 Rule found by C4.5H on the $D_4$(FC) homology dataset . . . . . 86
4.8 Rule found by CLUS-HMC on the $D_4$(FC) homology dataset . . . 87

4.9 Class-specific AUROC comparison between Clus-HMC-Ens and BSVM . . . . . 88

5.1 A maximum common subgraph of two molecular graphs . . . . . 102
5.2 Examples of molecular graphs and their MCS . . . . . 105
5.3 An outerplanar graph $G$ with some example BPSs and BSSs (1) . 108
5.4 An outerplanar graph $G$ with some example BPSs and BSSs (2) . 109
5.5 An outerplanar graph $G$ with some example BPSs and BSSs (3) . 110
5.6 Parent-child relationships between BPSs and BSSs . . . . . 113
5.7 An example matching between two BPSs $G^r$ and $H^s$ . . . . . 118
5.8 An example matching between two BSSs $G|^*_{o^G[r',r[}$ and $H|^*_{o^H[s',s[}$ . 120
5.9 Runtimes of $\text{MCS}_{\sqsubseteq}$ and $\text{MCS}_{\preceq}$ (in seconds) . . . . . 126

6.1 An MCS computed under the BBP subgraph isomorphism . . . . . 134
6.2 Predictive performance of the IBL-$d_{\sqsubseteq}$ and IBL-$d_{\preceq}$ classifiers . . . . 138
6.3 Predictive performance of the state-of-the-art IBL classifiers . . . . 139
6.4 Comparison of the performance of the SVM classifiers . . . . . 141

7.1 Predictive performance of different selection strategies . . . . . 153
7.2 Predictive performance when applying different constraints . . . . 155
7.3 Relationship between the number of pairwise comparisons and the number of unique MCSs . . . . . 159
7.4 Predictive performance of state-of-the-art feature generation methods 160

# List of Tables


2.1 Example of an attribute-value representation for molecules . . . . . 14
2.2 Example of a relational database representing molecules . . . . . 16
2.3 Representation of molecules in ILP . . . . . 21
2.4 Example of a learned rule in ILP . . . . . 24
2.5 Example of an alternative attribute-value representation of molecules 25
2.6 Contingency matrix for a binary classifier . . . . . 30

3.1 Comparing the three decision tree approaches at a conceptual level 49
3.2 Datasets used in the experimental study . . . . . 56
3.3 Properties of the two classification schemes . . . . . 57
3.4 Weighting schemes for FunCat and GO on the three PR measures 60
3.5 Comparing weighting schemes for FunCat . . . . . 60
3.6 Predictive performance of the different algorithms for FunCat . . . 62
3.7 Predictive performance of the different algorithms for GO . . . . . 62
3.8 Clus-HMC compared to Clus-SC and Clus-HSC (PR) . . . . . 63
3.9 Clus-HSC compared to Clus-SC (PR) . . . . . 63
3.10 Clus-SC, Clus-HSC and Clus-HMC compared to default (PR) 65
3.11 Difference between training set AUPRC and test set AUPRC . . . 66
3.12 Tree size (number of tree leaves) for FunCat . . . . . 69
3.13 Tree size (number of tree leaves) for GO . . . . . 70

4.1 AU($\overline{\text{PRC}}$) of Clus-HMC-Ens and the MouseFunc systems . . . . 91
4.2 $\overline{\text{AUPRC}}$ of Clus-HMC-Ens and the MouseFunc systems . . . . . 92
4.3 $\overline{\text{AUROC}}$ of Clus-HMC-Ens and the MouseFunc systems . . . . . 93

5.1 Distribution of runtimes of the two MCS algorithms . . . . . 127

6.1 Average AUROC and ranks for the state-of-the-art IBL classifiers . 140

7.1 Percentage of non-outerplanar examples in the datasets . . . . . 149
7.2 Overview of the different parametric choices for $\mathcal{F}$ . . . . . 151

7.3 Average AUROC and ranks for the different selection strategies . . 154
7.4 AUROC for the three classification tasks of the HIV dataset . . . . 154
7.5 Average AUROC and ranks when applying different constraints . . 156
7.6 Redundancy evaluation of the different feature construction methods 157
7.7 Distribution of the number of edges of three feature generation methods . . . . . 157
7.8 Predictive performance for an increasing number of randomly sampled MCSs . . . . . 158
7.9 Average AUROC and ranks for the state-of-the-art methods . . . . 161
7.10 AUROC of A-MCS on the classification tasks of PTC and Bursi . 162

# List of Algorithms


2.1 A generic TDIDT algorithm . . . . . 29
3.1 The top-down induction algorithm for PCTs . . . . . 44
4.1 Clus-HMC-Ens: the bagging algorithm around Clus-HMC . . . 78
5.1 Computing an MCS of two outerplanar graphs . . . . . 115
5.2 Computing an MCS of two block-preserving-subgraphs . . . . . 117
5.3 Computing an MCS of two block-splitting-subgraphs . . . . . 119

# Introduction

# Chapter 1

# Outline

In this chapter, we present an overview of the work in this thesis and motivate it within its research field. We start with discussing the context of this thesis in Sect. 1.1. Section 1.2 gives a motivation and lists the contributions of the thesis. Finally, we explain the structure of the rest of the thesis (Sect. 1.3) and present a short bibliography (Sect. 1.4).

## 1.1 Context

In this section, we briefly discuss the most important scientific fields related to this thesis.

### 1.1.1 Artificial intelligence

Artificial intelligence (AI) is the research field that has existed for just over 50 years, when John McCarthy organised the first conference dedicated to it. One the goals of this conference was to elaborate on the conjecture that every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it [McCarthy et al., 1955]. This has led to the following definition:

**Definition 1.1 (Artificial intelligence)** *Artificial intelligence is the science and engineering of making intelligent machines, especially intelligent computer programs.*

The enormous optimism in AI of those days and the subsequent attention it received in popular culture have led to huge expectations and quite often to misconceptions among the general public.

Nowadays, AI scientists have reformulated their research goals and, rather than trying to invent the ultimate AI machine that, just like humans, is able to solve any given problem, interact with various environments and adapt to new situations, they have worked towards practical solutions for well-specified problems in AI.

The question "What is artificial intelligence?" is still very difficult to answer though. One of the reasons is that we are barely beginning to understand what *human* intelligence is. For example, is a human able to compute huge mathematical calculations considered as intelligent? Is the ability to play chess the result of human intelligence? And what about being able to navigate in an unknown environment? Most people would consider an entity able to solve all these problems as being intelligent.

Not unexpectedly, human intelligence is not equal to the intelligence that can be displayed by a computer. Being able to perform huge calculations quickly, which is a challenge for the human mind, is not considered to be hard for a computer equipped with a reasonable processor and sufficient memory. Conversely, tasks that are considered easy for humans can be very challenging for a computer (program) e.g., recognising human emotions or walking on two legs.

Most of the tasks mentioned above turn out to be well-specified AI problems. For a lot of these problems, AI programs have been developed that are better at solving them than most humans. A famous example is the defeat of the chess world champion Garri Kasparov against the *Deep Blue* chess computer developed by IBM in 1997. In everyday life, AI has proved its usefulness in applications such as spam filtering of e-mails or speech recognition. Also the computer programs that recommend new friends in a social network like Facebook or advise to buy specific books based on the ones already purchased are the result of AI research.

By focusing on particular problems, the scientific domain of artificial intelligence nowadays covers multiple research areas, such as knowledge representation, reasoning, planning, natural language processing and computer vision. In this thesis, we will focus on machine learning, which is considered as one of the key elements of artificial intelligence.

### 1.1.2 Machine learning

Machine learning is the subfield of artificial intelligence that is concerned with the development of computer programs that have the ability to learn. Mitchell [1997] formally defines learning as follows:

**Definition 1.2 (Machine learning)** *A computer program is said to learn from experience $E$ with respect to some class of tasks $T$ and performance measure $P$, if its performance at tasks in $T$, as measured by $P$, improves with experience $E$.*

We explain this definition with an example from chemoinformatics.

**Example 1.1** *An interesting task in drug development is predicting a molecule's toxicity. The performance of a computer program tackling this task can be measured by the percentage of molecules that are correctly predicted. The experience of the program is the number of molecules with known toxicity that it has already observed. If the program improves its predictions by observing more examples, it is then considered to have the ability to learn.*

A learning algorithm takes as input a set of examples and returns a model that can be used for predictive and descriptive purposes:

- The objective of **predictive learning** is to obtain models that predict a target variable for unseen examples. For instance, such models could predict whether an unknown molecule is toxic, or what the chances are that it is effective against some disease. Examples of predictive learning algorithms include decision trees, Bayesian networks and support vector machines [Witten and Frank, 2005].
- The objective of **descriptive learning** is to find patterns in the examples. For instance, apart from knowing whether a molecule is toxic, it is also interesting to find the particular fragments of the molecule that can explain its activity. Examples of descriptive learning algorithms include association rule mining, clustering and principal-component analysis [Bishop, 2006].

Traditional learning algorithms, called *attribute-value* learners, consider data that can be described by a fixed set of attributes or properties. However, many real-world applications come with data that have a far more complex structure, such as images, roadmaps, molecules or gene networks. In that case, the attribute-value setting is insufficient. For this reason, relational learning algorithms have been developed over the last years [Džeroski and Lavrač, 2001b]. These algorithms learn models that deal with structured data, such as relational databases, XML-documents, images, time series, sequences or graphs. There are two fields within relational learning in which we are particularly interested:

- **Structured input learning** concerns learning algorithms that have structured data as input. Example 1.1 deals with structured input learning as it considers molecules as input.
- **Structured output learning** concerns learning algorithms that produce models of which the output can be structured. For example, instead of predicting a single target variable, the output can consist of multiple variables, stored in for instance a vector or a graph.

Machine learning is an interdisciplinary research domain which uses techniques and concepts from artificial intelligence, statistics, probability theory and even biology. It is also closely related to data mining, which involves extracting knowledge from large databases.

### 1.1.3 Knowledge discovery and data mining

Data mining is a single step in the process of knowledge discovery in databases [Fayyad et al., 1996]:

**Definition 1.3 (Knowledge discovery in databases)** *Knowledge discovery in databases is the non-trivial process of identifying novel, valid patterns in data.*

The following keywords are important in this definition:

- *non-trivial:* the acquired knowledge should not be the result of applying simple statistics but some kind of search or inference should be involved;
- *novel:* the learned patterns should be new (and preferably unexpected) to the user and at the same time they should be understandable such that a domain expert can extract useful knowledge from them;
- *valid:* the patterns should be conform the user's specifications and hold against the given data.

The process of knowledge discovery consists of several steps: business understanding, data understanding, data preparation, data mining, evaluation and deployment. Data mining, which is concerned with applying techniques from, for instance the machine learning domain, is the central step in this process. The reason why it has become important is twofold. First, over the last few years, many techniques have been developed that can generate enormous amounts of data. For example, intensive care units in hospitals nowadays use all kinds of sensors, measuring the heart rate or blood pressure, to monitor the condition of a patient, or department stores keep track of all purchases made by their customers. Second, the technological advances in database technology allow to store huge amounts of data and to easily access it. Data mining has received a lot of attention in the scientific and industrial community, with applications in marketing, fraud detection and signal processing.

Relational data mining is the specific subfield of data mining that applies relational learning algorithms. One type of relational data mining is graph mining, in which the data to be mined consist of graphs.

**Example 1.2** *Consider a dataset of molecules represented as graphs. A possible data mining task is to find the particular molecule fragments that are responsible for the activity that is measured in the dataset. A technique that can be used for this is frequent subgraph mining. It enumerates all subgraphs that occur a certain number of times in the dataset. From this set, the subgraphs that correlate most with the activity can be selected, which can then be investigated by chemists.*

### 1.1.4 Biological applications of learning and mining

When Watson and Crick [1953] discovered the double helix structure of DNA, which carries the genetic information of a living organism, a revolution in biology began, creating a new research field called computational biology. Watson and Crick showed that the genetic code of every living being on this planet is quadruple: it consists of only four different nucleotides and it is actually not that different from the binary code that is used to store information and programs in a computer.

The central dogma of molecular biology states that the flow of information is passed from DNA to proteins. Proteins are considered to be the workhorses of the body. They are responsible for many important functions, such as catalysation, transport, cell signalling, gene regulation and structural support. During the last decade, many biologists hoped that, if they were to unravel the genetic code or DNA of these proteins, they would gain a lot of insight in the way cells work and that enormous advances would be possible in the way diseases are treated. However, the translation of the genetic code has turned out to be a very complex process. Although many insights have been acquired in various biological pathways, a lot of questions, for instance how a disease like Alzheimer's can develop, still remain open.

Since the completion of the *Human Genome Project*[1] in 2001, the goal of which was to determine the complete genome sequence of a human being, the amount of genetic information about many organisms has been doubling every 18 months, somewhat similar to the law of Moore, who predicted a similar increase in computer processor power. Apart from techniques that determine genomic data, there exist a huge number of high-throughput techniques that are able to generate all kinds of other data about genes and their activity in the cell. For example, microarrays measure the expression level of thousands of genes simultaneously in a single experiment. Similar techniques are able to generate data like transcriptomics, metabolomics and pharmacogenomics and all these data are stored in public databases like Entrez, GenBank or UniProt. The availability of these data creates a lot of opportunities for machine learning and data mining [Page and Craven, 2003], which can be used to tackle tasks like predicting gene function, finding pathways in protein networks or predicting resistance of HIV. However, as the data involved in these tasks usually have a complex structure, traditional data mining techniques often fall short, and more advanced techniques such as the ones of relational data mining are required.

**Example 1.3** *The goal of pharmaceutical companies is to develop new drugs. An important step in this process is the identification of molecules that play an active role in the regulation of the biological process or disease state under consideration. This step is called screening and it is costly and time-consuming, since hundreds of thousands of molecules are possible candidates for a new drug. In the past,*

[1] `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`

*chemists tested molecules in the lab, but nowadays, pharmaceutical companies rely on virtual techniques, which can automatically select a limited number of candidate molecules, in order to reduce the amount of molecules that should be tested in the lab. However, as molecules have a structure, i.e. there is a relationship between their atoms and bonds, no techniques exist that can cope with them efficiently without ignoring the structural information.*

## 1.2 Motivation and contributions

Because of the many high-throughput techniques that can easily generate biological data, there is a need for efficient techniques that can extract useful knowledge from biological data. Relational learning and data mining techniques provide the strong expressiveness that is necessary to deal with biological tasks involving structured data, but often suffer from efficiency issues. The overall goal of this thesis is *to improve the efficiency of these methods, as well as their applicability to real-life applications from biology and chemistry.*

We will try to achieve this goal at two different levels. First, we will represent the biological data with graphs, as these have proved a promising alternative to more complex data structures such as logic programs. Molecules are a perfect example as their atom-bond structure matches the structure of a graph naturally. But also other biological data can be easily represented by graphs, such as the secondary structure of mRNA or networks of gene interactions. Moreover, as graphs and their properties have been thoroughly studied in the field of mathematics, much of this research can be used to improve learning algorithms. If certain properties in the graphs that represent biological data can be exploited, more efficient algorithms can be developed. For example, graphs that represent molecules are known to have a bounded degree or are planar.

Second, we will align the learning methods better with the biological task under consideration. For example in gene function prediction, multiple classes need to be predicted, and these classes have a relationship between each other, so it makes sense to develop learning methods that take into account these relationships.

The main contributions of this thesis can be summarised as follows. In the first part of the thesis, we focus on the application of functional genomics. Here, the task is to predict the function of unknown genes found in the genome of an organism. It is known that a gene may have multiple functions. Moreover, biologists have organised these functions into hierarchies. This setting is known in machine learning as hierarchical multi-label classification (HMC). It is a variant of classification where instances may belong to multiple classes at the same time and where these classes are organised in a hierarchy. In this context, we focus on decision trees because of their interpretability, since we believe it is important for biologists to find out why certain functions are predicted.

- The **first contribution** is the introduction of three different learning approaches for decision trees in the context of HMC, as well as an empirical study of their use in functional genomics. We compare learning a single HMC tree (which makes predictions for all classes together) to two approaches that learn a set of regular classification trees (one for each class). The first approach defines an independent single-label classification task for each class (SC). Obviously, the hierarchy introduces dependencies between the classes. While they are ignored by the first approach, they are exploited by the second approach, named hierarchical single-label classification (HSC). We compare the three approaches on 24 yeast datasets using as classification schemes MIPS's FunCat (tree structure) and the Gene Ontology (DAG structure). We show that HMC trees outperform HSC and SC trees along three dimensions: predictive performance, model size, and induction time.

- The **second contribution** is an extensive comparison of the proposed HMC decision tree method to state-of-the-art methods for functional genomics. We show that our HMC trees obtain clearly better predictive performance than the trees found by previously proposed decision tree methods. Moreover, we also introduce ensembles of HMC trees, which obtain better predictive performance than single trees and are competitive with statistical learning and functional linkage methods. Moreover, the ensemble method is computationally efficient and easy to use.

In the second part of the thesis, we focus on the application of learning structure-activity relationships (SAR), where the task is to predict properties of molecules based on their atom-bond structure. Since we want to preserve the structural information of the molecules, we will represent them as graphs.

- The **third contribution** involves the introduction of a polynomial algorithm to compute a maximum common subgraph (MCS) of two outerplanar graphs. By using the BBP subgraph isomorphism, which is especially designed for use in the SAR context, we show that this algorithm is significantly faster than a state-of-the-art MCS algorithm using the general subgraph isomorphism.

- The **fourth contribution** concerns a metric for structured data that is based on the proposed MCS algorithm. We evaluate the metric as a similarity measure for molecules and we show that it obtains state-of-the-art results for several SAR tasks. More generally, we show that using the BBP subgraph isomorphism as matching operator improves the predictive performance of graph mining methods.

- The **fifth contribution** is the application of the MCS algorithm to feature generation. Rather than mining for all optimal local patterns, we sample features from the set of pairwise maximum common subgraphs. We show

that using simple sampling strategies we obtain significant gains in speed while at the same time improving the quality of the extracted features. We observe a significant increase in predictive performance when using maximum common subgraph features instead of frequent or correlated local patterns on 60 benchmark datasets from NCI. Moreover, with a much smaller set of features, it is possible to reach the same predictive performance as methods that exhaustively enumerate all possible patterns.

## 1.3 Structure of the text

In this chapter, we have briefly outlined the context of this thesis and we have summed up its contributions. **Chapter 2** introduces the background and the basic concepts that will be used in this thesis.

Then, we divide the research into two parts. The first part is concerned with hierarchical multi-label classification, for which the goal is to predict multiple classes in a given hierarchy. Since this corresponds to predicting one or more paths in the graph that represents the hierarchy, this can be seen as a structured output learning problem.

- **Chapter 3** contains a description of the proposed HMC decision tree method and introduces the empirical study of HMC decision tree learning.

- In **Chapter 4**, we introduce the ensemble method of our HMC trees and compare it to state-of-the art methods found in the biomedical literature.

The second part is concerned with structure-activity learning. By representing the molecules as graphs and by exploiting the properties of these molecular graphs, efficient algorithms can be developed. The methods developed in this context belong to the structured input learning setting, and more particularly to graph mining.

- In **Chapter 5**, we propose the polynomial algorithm to compute a maximum common subgraph (MCS) of two outerplanar graphs.

- **Chapter 6** presents the efficiently computable metric for outerplanar graphs that is based on the MCS algorithm. The properties of this metric are empirically investigated on a number of molecular datasets.

- **Chapter 7** contains the feature generation method for graphs that computes maximum common subgraphs from randomly selected pairs of examples.

Finally, **Chapter 8** summarises the main conclusions of this thesis and suggests some possible directions for further research.

## 1.4 Bibliographical note

Most parts of this thesis have been published before. The following list contains the key articles. A complete publication list of the author can be found at the end of this text (page 193).

- **Chapter 3: Decision trees for hierarchical multi-label classification**
  - Hendrik Blockeel, Leander Schietgat, Jan Struyf, Sašo Džeroski, and Amanda Clare, *Decision trees for hierarchical multi-label classification: A case study in functional genomics*, European Conference on Principles and Practice of Knowledge Discovery, Lecture Notes in Computer Science, volume 4213, pages 18-29, 2006.
  - Celine Vens, Jan Struyf, Leander Schietgat, Sašo Džeroski, and Hendrik Blockeel, *Decision trees for hierarchical multi-label classification*, Machine Learning, volume 73, issue 2, pages 185-214, 2008.
- **Chapter 4: Decision tree ensembles for gene function prediction**
  - Leander Schietgat, Celine Vens, Jan Struyf, Hendrik Blockeel, Dragi Kocev, and Sašo Džeroski, *Predicting gene function using hierarchical multi-label decision tree ensembles*, BMC Bioinformatics, volume 11, issue 2, 2010.
- **Chapter 5: A polynomial-time maximum common subgraph algorithm**
  - Leander Schietgat, Jan Ramon, and Maurice Bruynooghe, *A polynomial-time metric for outerplanar graphs*, Machine Learning Conference of Belgium and the Netherlands, pages 97-104, 2007.
- **Chapter 6: An efficiently computable metric for outerplanar graphs**
  - Leander Schietgat, Jan Ramon, Maurice Bruynooghe, and Hendrik Blockeel, *An efficiently computable graph-based metric for the classification of small molecules*, International Conference on Discovery Science, Lecture Notes in Computer Science, volume 5255, pages 197-209, 2008.
- **Chapter 7: Maximum common subgraph sampling**
  - Leander Schietgat, Fabrizio Costa, Jan Ramon, and Luc De Raedt, *Effective feature construction by maximum common subgraph sampling*, Machine Learning, to appear, 2010.

# Chapter 2

# Background

## 2.1 Introduction

This chapter gives a general introduction to the basic concepts that will be used in the rest of this thesis. For a more detailed discussion about concepts and techniques from machine learning and data mining, we refer to [Mitchell, 1997; Blockeel, 2007]. Throughout this chapter, we will give additional references for further reading.

The chapter is organised as follows. In Sect. 2.2, we explain the characteristics of structured data. Section 2.3 discusses the most important learning settings, and describes their advantages and shortcomings. We will use the representation of molecules in the different settings as an illustrating example. Because we will represent molecules with graphs in the second part of the thesis, we will have particular interest in graph mining techniques. In Sect. 2.4, we focus on decision tree learning, since decision trees form an important component of the first part of this thesis. Finally, we discuss several measures to evaluate the quality of the learning algorithms in Sect. 2.5.

## 2.2 Structured data

In machine learning and data mining, the term *structured data* is used to indicate that some kind of relational structure is present in the data. For example, molecules have atoms and bonds, which indicate how the atoms are linked to each other. There are several ways to represent structured data, such as relational databases, graphs, logic programs or any kind of knowledge representation that is capable to define relationships in the data. In this section, we discuss several possible representations. However, we first explain the counterpart of structured

Table 2.1: Example of an attribute-value representation for molecules.

| Molecule ID | Size | Mass | Charge | Polarity | Toxicity |
|---|---|---|---|---|---|
| 1 | large | 180 | neutral | polar | no |
| 2 | small | 57 | positive | apolar | no |
| 3 | midsize | 125 | negative | polar | yes |
| 4 | small | 33 | neutral | apolar | no |
| 5 | midsize | 85 | positive | polar | yes |

data, namely attribute-value data, which is still the most widely used data representation in machine learning and data mining algorithms.

### 2.2.1 Attribute-value data

Attribute-value data are data that can be stored in a single table. Every row of the table represents an observation (called *example*), while every column represents a property (called *attribute*) of these observations. An example can be represented as an element of an instance space $\mathcal{I}$, where $\mathcal{I}$ is the product set of a number of attribute domains $A_i$, that is, $\mathcal{I} = A_1 \times A_2 \times \ldots \times A_n$, with $n$ the number of attributes in the dataset. Each attribute domain $A_i$ can have several attribute values $a_i$, which can be of several types: they are called *nominal* if they consist of an unordered set of values, *ordinal* if there is an order in the set of values, and *numerical* if they take values from the set of real numbers. In predictive learning, the aim is to predict one of these attributes, called the target attribute.

**Example 2.1** *Table 2.1 shows a simplified example of an attribute-value data representation. Every row of the table corresponds to a molecule, which has five chemical properties: size, mass, charge, polarity and toxicity. Note that "Molecule ID" is a database key and will not be used as an attribute for learning. The attribute size is ordinal because its values can be ordered from large to small. The attribute mass is numerical, while charge, polarity and toxicity are nominal attributes, as there is no ordering between their values. If a nominal attribute has only two possible values, which is the case for polarity and toxicity, it is also called boolean. A possible learning task for this dataset is to predict whether a molecule is toxic (by specifying toxicity as the target attribute). A learning model will then be based on the values of the four other attributes.*

The attribute-value data representation is very popular because it serves as input for a lot of efficient off-the-shelf learning algorithms, such as decision trees, support vector machines, Bayesian networks or rule sets. However, the attribute-value format is not suitable for most data from real-world applications. For example, only general properties of a molecule are reported in Table 2.1. These give

no direct indication of the particular atom-bond structure of the molecules, while this structure is known to be important when predicting their activity. It is not straightforward to represent the atom-bond structure of molecules in a single table. First, the number of atoms and edges in a molecule is variable, while a single table requires a fixed number of attributes. Second, since an edge can occur between every pair of vertices, an attribute should be reserved for every combination of two atoms, indicating whether a bond between them exists. This would lead to a huge number of attributes [De Raedt, 1998]. The process of transforming structured data, such as the atom-bond structure of molecules, into a fixed-attribute table is almost a research domain on its own and will be discussed Sect. 2.3.3.

### 2.2.2 Relational databases

A relational database consists of several tables that are linked to each other [Elmasri and Navathe, 2004]. Each table in a relational database represents either an entity or a relation between entities. The attributes in these tables describe properties of the entities or relations. The subset of attributes with which one can uniquely identify the tuple in the table are called the key attributes (these are usually underlined in the tables). The keys are also used to relate the different tables to each other.

**Example 2.2** *The Mutagenesis dataset [Srinivasan et al., 1996] describes the atom-bond structure of 230 molecules. Each molecule consists of a number of atoms that can be bound to each other. An atom is characterised by its element (e.g., carbon, nitrogen), its type (e.g., aromatic carbon, aryl carbon), and its partial charge. Bonds between atoms also have a type (e.g., single, double, aromatic). Table 2.2 shows two tables for the entities* Molecule *and* Atom*, and one for the relation* Bond*. The target attribute is mutagenicity, which indicates whether the molecule can cause damage to DNA and lead to cancer.*

Relational databases provide a lot of expressiveness compared to a single table in the attribute-value format, but they are much too general than what is needed to, for example, represent molecules.

### 2.2.3 Graphs

In mathematics, a graph is an abstract representation of a set of objects and their relationships. The objects are called *vertices*, and the links that connect pairs of vertices are called *edges*. Typically, a graph is depicted in diagrammatic form as a set of dots for the vertices, joined by lines or curves for the edges. More formally, a graph is defined as follows.

**Definition 2.1** *A graph is a tuple $G(V, E)$, with $V$ a finite set of vertices and $E \subseteq \{(u, v) \mid u, v \in V\}$ a set of edges.*

Table 2.2: Example of a relational database representing molecules.

MOLECULE

| **Molecule ID** | **Mutagenicity** |
|---|---|
| 1 | yes |
| 2 | yes |
| 3 | no |
| ... | |

ATOM

| **Atom ID** | **Molecule ID** | **Element** | **Type** | **Charge** |
|---|---|---|---|---|
| 1 | 1 | carbon | 22 | -0.117 |
| 2 | 1 | carbon | 22 | -0.117 |
| 3 | 1 | carbon | 22 | -0.117 |
| 4 | 1 | carbon | 195 | -0.087 |
| 5 | 1 | carbon | 195 | 0.013 |
| 6 | 1 | carbon | 22 | -0.117 |
| 7 | 1 | hydrogen | 3 | 0.142 |
| ... | | | | |

BOND

| **Molecule ID** | **Atom ID1** | **Atom ID2** | **Type** |
|---|---|---|---|
| 1 | 1 | 2 | 7 |
| 1 | 2 | 3 | 7 |
| 1 | 3 | 4 | 7 |
| 1 | 4 | 5 | 7 |
| 1 | 5 | 6 | 7 |
| 1 | 6 | 1 | 7 |
| 1 | 1 | 7 | 1 |
| ... | | | |

If $G$ is a graph, we denote with $V(G)$ the set of vertices of $G$, with $E(G)$ the set of edges of $G$. A graph is called:

- **undirected** if edges $(u, v)$ and $(v, u)$ are not distinguished. Otherwise a graph is directed. An edge of the form $(v, v)$ is called a loop;
- a **multigraph** if the set of edges $E$ is a multi-set;
- **connected** if there is a path between every two vertices;
- **labeled** if there exists a finite set of labels $\Sigma$ and a labelling function $\lambda$ :

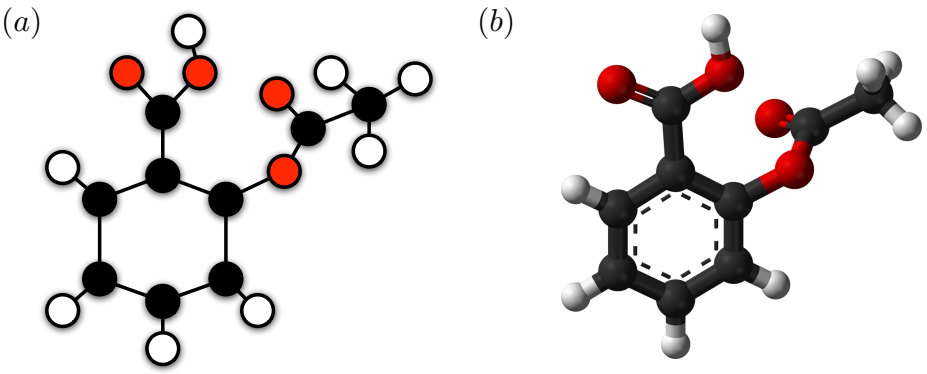


Figure 2.1: Two representations of the molecule aspirin: (a) graph structure (b) 3D structure.

$$V \cup E \rightarrow \Sigma.$$

We will denote the set of all graphs with $\mathcal{G}$. In this text, we will mostly consider undirected, labeled graphs $G(V, E, \lambda, \Sigma)$, such as the one in Fig. 2.1a.

Graphs can be used to represent several types of relational data. In this thesis, we will mainly use them as a representation for molecules, but there are many different examples, such as roadmaps (in which vertices are crossroads and edges are roads), social networks (in which vertices are people and edges represent the friend relationship) or protein-protein interaction networks (in which vertices are proteins and edges their interactions).

**Example 2.3** *The graph in Fig. 2.1a is a representation of the molecule aspirin, for which the 3D structure is depicted in Fig. 2.1b. In Fig. 2.1a, colours are used for the node labels, which indicate the chemical element: black for carbon, red for oxygen and white for hydrogen. Note that we do not show edge labels in this example.*

**Example 2.4** *Figure 2.2 shows an example of the Gene Ontology [Ashburner et al., 2000], which is a hierarchy of gene functions that is developed by biologists. The graph that represents this ontology consists of directed edges, for which the arrows determine their direction. Moreover, there are no directed cycles, that is, there is no way to start at some vertex $v$ and follow a sequence of edges that eventually loops back to $v$ again. Such graphs are called directed acyclic graphs.*

The advantages of graphs are that they can naturally represent relational data and that they are intuitive because of their visual nature. However, the increased expressivity compared to attribute-value data comes at the cost of efficiency: a

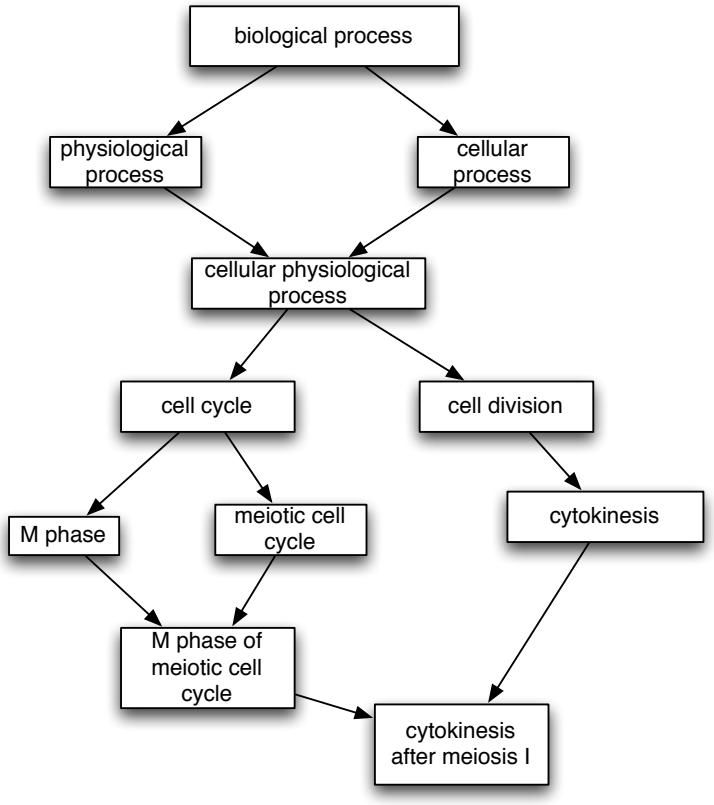


Figure 2.2: Example of a part of the Gene Ontology represented as a directed acyclic graph.

lot of operations on graphs are computationally expensive. As an example, we introduce the subgraph isomorphism, which is known to be NP-complete [Garey and Johnson, 1979].

**Definition 2.2** *Two graphs $G$ and $H$ are isomorphic if there exists a bijection $\varphi : V(G) \to V(H)$ such that $\forall u, v \in V(G)$ the following holds: (i) $\{u, v\} \in E(G) \Leftrightarrow \{\varphi(u), \varphi(v)\} \in E(H)$, (ii) $\lambda_G(u) = \lambda_H(\varphi(u))$, and (iii) $\{u, v\} \in E(G) \Rightarrow \lambda_G(\{u, v\}) = \lambda_H(\{\varphi(u), \varphi(v)\})$.*

**Definition 2.3** *Let $G$ and $H$ be graphs. $G$ is a subgraph of $H$, if (i) $V(G) \subseteq V(H)$, (ii) $E(G) \subseteq E(H)$, and (iii) $\lambda_G(x) = \lambda_H(x)$ holds for every $x \in V(G) \cup E(G)$.*

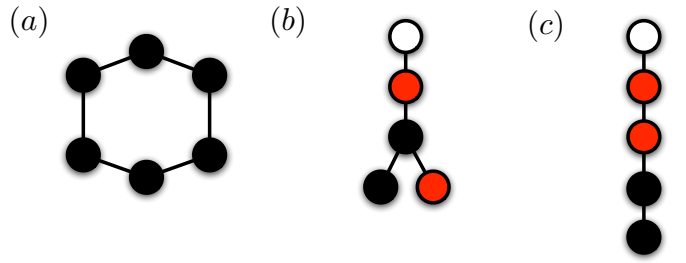


Figure 2.3: Molecular fragments represented as graphs.

**Definition 2.4** *A graph $G$ is subgraph isomorphic to $H$ iff $G$ is isomorphic to a subgraph of $H$.*

Intuitively, the subgraph isomorphism checks whether a graph $G$ is a part of another graph $H$. If it is, we say that $G$ can be embedded in $H$. Figure 2.3 shows three graphs which could be representations of molecular fragments. The graphs in Fig. 2.3a and Fig. 2.3b can be embedded in the graph representing aspirin (see Fig. 2.1a), while the graph in Fig. 2.3c cannot.

Graph theory has been an active research field for several centuries and their properties have been thoroughly investigated. For example, classes of graphs that are well-known in the computer science domain are trees (which are connected graphs for which $|V(G)| = |E(G)| + 1$) or sequences (which are trees for which every vertex has at most two edges). We denote the set of all trees with $\mathcal{T}$ and the set of all sequences with $\mathcal{S}$.

**Example 2.5** *Figure 2.3b shows an example of a tree, while Figure 2.3c is an example of a sequence.*

Because sequences and trees have a more restricted structure than general graphs, the subgraph isomorphism can be computed for these in polynomial time [Garey and Johnson, 1979].

As Fig. 2.4 shows, the several classes of graphs form subsets of each other. The set of sequences is a subset of the set of trees, which is in turn a subset of the set of general graphs. There exist a lot of graph classes $\mathcal{T}$ beyond sequences and trees, which are still more specific than general graphs. These are of particular interest to us since a possible strategy to improve the efficiency of graph mining techniques consists of looking for specific properties of particular graph classes and exploiting these properties [Horváth et al., 2006; Horváth and Ramon, 2008]. The idea is to find graph classes that are able to represent molecules and for which problems as the subgraph isomorphism are still efficiently computable.

If the concept of a graph is generalised to a hypergraph, where an edge can connect any number of vertices, there is a clear relationship with relational databases:

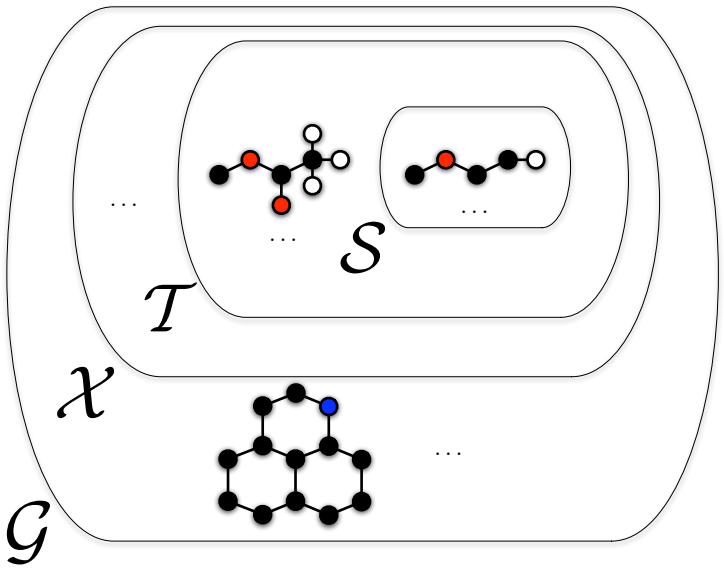


Figure 2.4: The hierarchy of graphs.

each tuple $(v_1, v_2, \ldots, v_n, l_1, l_2, \ldots, l_m)$ in a relation $R$ then corresponds to a hyperedge $(v_1, v_2, \ldots, v_n)$ with labels $(l_1, l_2, \ldots, l_m)$.

For a more detailed overview of graph theory, we refer to the book of Diestel [2000].

### 2.2.4 Logic programs

The use of first order logic as a representation for structured data has become popular in the context of inductive logic programming. In ILP, the data as well as the models are represented as logic programs. The programming language Prolog is used to this end [Bratko, 2001].

First, we introduce some terminology. A term is a constant, a variable or a function symbol immediately followed by a tuple of terms. The length of the tuple is called the arity of the term. An atom consists of a predicate symbol and a tuple of terms; a literal is either an atom (positive literal) or its negation (negative literal). A clause is a disjunction of positive or negative literals, and are often represented as a rule, where the positive literals occur in the head and the negative ones in the body of the rule. A set of clauses with one literal in the head is called a predicate definition. A logic program is then a set of predicate definitions.

A predicate definition can be defined extensionally or intensionally. An ex-

Table 2.3: Representation of molecules in ILP.

```
molecule(m1, mutagenic).                bond(m1,a1,a2,7).
atom(m1,a1,c,22,-0.117).                bond(m1,a2,a3,7).
atom(m1,a2,c,22,-0.117).                bond(m1,a3,a4,7).
atom(m1,a3,c,22,-0.117).                bond(m1,a4,a5,7).
atom(m1,a4,c,195,-0.087).               bond(m1,a5,a6,7).
atom(m1,a5,c,195,0.013).                bond(m1,a6,a1,7).
atom(m1,a6,c,22,-0.117).                bond(m1,a1,a7,1).
atom(m1,a7,h,3,0.142).                  bond(m1,a2,a8,1).
...                                     ...
molecule(m2, nonmutagenic).
...
% Background knowledge
sbond(M,A1,A2,T) :- bond(M,A1,A2,T).
sbond(M,A1,A2,T) :- bond(M,A2,A1,T).
```

tensional predicate definition enumerates all objects for which the predicate holds. This means the predicate definition consists of a set of ground facts, that is, clauses with only one positive literal that does not contain variables. An intensional predicate definition on the other hand, defines the predicate in terms of other predicates. Very often the extensionally defined predicates definitions are referred to as the database and the intensionally defined predicates definitions as the *background knowledge*. The advantage of background knowledge is that new ground facts can easily be deduced from it without having to enumerate them.

**Example 2.6** *An example of a logic representation of a set of molecules from the Mutagenesis dataset can be found in Table 2.3. The first part consists of ground facts of the predicates* `molecule/2`*,* `atom/5` *and* `bond/4`*. The predicate* `molecule/2`*, with attributes MolID and Mutagenic, corresponds to the* MOLECULE *entity. This predicate will be the target predicate, for which MolID is the key variable and Mutagenic is the target variable. In the same way, the predicate* `atom/5`*, with attributes MolId, AtomId, Element, Type and Charge, corresponds to the* ATOM *entity and the predicate* `bond/4`*, with attributes MolId, AtomId1, AtomId2 and Type, corresponds to the* BOND *relation. The second part consists of background knowledge. The relation* BOND *is actually symmetric: if* `bond(M,A1,A2,T)` *holds, then* `bond(M,A2,A1,T)` *must hold as well, as bonds are undirected. The use of background knowledge enables us to specify this symmetry without having to enumerate the two directions of every bond.*

As can be noticed from Table 2.2 and Table 2.3, there is a clear relationship

between a logic program and a relational database: every table of the relational database corresponds to a predicate, while the attributes correspond to the arguments of the table. For more information about logic programming, we refer to [Bratko, 2001].

## 2.3 Types of learning and mining

The goal of machine learning and data mining is to learn models from data. A model is a piece of knowledge that provides an abstract description of (a subset of) the data. It can be used for predictive or descriptive purposes.

In *predictive learning*, the model makes predictions for unseen examples. When the target attribute is nominal, the learning process is called *classification*. Mostly, the term classification is used for the specific case of binary classification, where the target attribute is boolean. If the target attribute has more than two values, we speak of multi-class classification. When the target attribute is ordinal, we speak of *ranking* and when it is numerical, it is called *regression*. If there are multiple target attributes, we call this setting multi-target prediction. Examples of predictive learning algorithms include decision trees, Bayesian networks and support vector machines [Witten and Frank, 2005].

In *descriptive learning*, the model describes regularities in the data. So it is not necessarily a function predicting a target attribute. For example, association rules describe dependencies between certain attributes, while clusters group similar instances together. Examples of descriptive learning algorithms include association rule mining, clustering and principal-component analysis [Bishop, 2006].

Learning algorithms can be propositional or relational, which we will discuss in the next sections.

### 2.3.1 Attribute-value learning

Attribute-value learning uses attribute-value data as input and is sometimes referred to as propositional learning, because of its close relationship to propositional logic. It is the most widely used approach in machine learning and data mining.

As already mentioned, the advantage of attribute-value learning is that a lot of efficient learning techniques can be applied out of the box. However, in many real-world applications, the data is much more complex and cannot be stored in a single table. Apart from the difficulties with representing the variable atoms and edges of molecules in a single table, there are other problems.

For example, it is known that a molecule can have multiple conformations. A conformation is the structural arrangement of a molecule's atoms in three-dimensional space. Depending on the environment in which the molecule resides, it has a particular conformation. If one wants to predict for example a molecule's

toxicity, it is important to consider *all* of its possible conformations. If one of those conformations is toxic, then it is not advisable to use the molecule as a drug.

It is not straightforward to represent the multiple conformations of a single molecule into one row of the table. One of the difficulties is that the number of conformations can be different for each molecule, while the number of attributes in the propositional setting is fixed. The above setting has been studied in machine learning as multi-instance learning [Dietterich et al., 1997].

### 2.3.2 Relational learning

Because of the limitations of attribute-value learning, more and more methods that can handle relational data have been proposed, leading to the field of relational learning [Džeroski and Lavrač, 2001b; De Raedt, 2008]. Here, we will focus on two specific formalisms in relational learning: inductive logic programming (Sect. 2.3.2.1) and graph mining (Sect. 2.3.2.2).

A special distinction within relational learning is made w.r.t. the input and output of the learned models. In *structured input learning*, the input is structured, which is for example the case when using the atom-bond structure of molecules. In *structured output learning*, the output of the learning algorithm is structured. The structure can be fixed or variable. An example of the former case is predicting a vector of fixed length, which can can occur if one wants to predict the toxicity of molecules in different environments at once. An example of the latter case is, instead of predicting the activity of molecules, predicting the structure of the molecule that is most active.

#### 2.3.2.1 Inductive logic programming

*Inductive logic programming* (ILP) uses the formalism of first-order logic to represent the input data on the one hand and to represent the hypothesis on the other hand. Moreover, it is possible to define background knowledge. Because of these possibilities, ILP provides a high expressiveness with which complex patterns can be found. Table 2.4 shows an example rule that could be learned by an ILP system. The rule consists of three tests, checking whether a carbon and a nitrogen occurs in the molecule, and whether the nitrogen atom is positively charged. Because the charge is numerical, the predicate `atom_charge/2` is used to make the value discrete.

ILP also enables the incorporation of probabilities easily [De Raedt et al., 2007]. Many propositional learning techniques have been upgraded to the relational case in the context of ILP, such as decision tree learning [Blockeel and De Raedt, 1998] or mining association rules [Dehaspe and Toivonen, 1999; King et al., 2001].

A drawback of ILP is that a lot of problems are undecidable in first-order logic. Still, in spite of these computational challenges, ILP has known many successful biological applications [Srinivasan et al., 1997; Page and Craven, 2003; King, 2004].

Table 2.4: Example of a learned rule in ILP.

```
mutagenic(X) :-
      atom(X,X1,c,_,_),
      atom(X,X2,n,_,_),
      atom_charge(X2,positive).
```

More information about inductive logic programming can be found in [Muggleton and De Raedt, 1994; De Raedt, 2008].

#### 2.3.2.2 Graph mining

In *graph mining*, the input data are represented as graphs. One of the key applications of graph mining is learning the structure-activity relationships in molecules. Over the last years, graph mining has proved to be a fitting alternative to ILP on this learning task, finding a good balance between expressiveness and efficiency.

There are two settings in graph mining. First, if every example in the dataset consists of a graph, this is called the *transactional setting* and the goal is for example to predict a property for each of the graphs. An example of this is the structure-activity learning of molecules. Second, in the *single network setting*, the input consists of one single graph, such as a protein-protein interaction network and the goal is to predict vertices or edges in the network. Another example is trying to predict vertices (corresponding to functions) in the Gene Ontology, of which a part was shown in Fig. 2.2.

Just as for ILP, propositional approaches have been upgraded to the graph setting as well, including techniques from pattern mining [Yan and Han, 2002; Nijssen and Kok, 2004; Chi et al., 2005] and kernel methods [Horváth et al., 2004; Ceroni et al., 2007; Shervashidze and Borgwardt, 2009]. The task of frequent subgraph mining, for example, enumerates all graphs that have a certain frequency w.r.t. the graphs in the dataset, where the frequency of a pattern is defined as the percentage of graphs in the dataset in which the pattern can be embedded.

A lot of algorithms for frequent subgraph mining have been proposed [Yan and Han, 2002; Nijssen and Kok, 2004; Kuramochi and Karypis, 2004b]. Many of them use ideas from the Apriori-algorithm [Agrawal et al., 1993] for finding frequent itemsets, which has been developed with the application of market basket analysis in mind. Pattern mining algorithms consist of two essential operators. The first is the enumeration of candidates, making sure that each candidate is selected only once (for graphs this means avoiding isomorphic copies), while the second is the frequency test, which checks the embedding of every pattern in the dataset examples. The NP-complete subgraph isomorphism is used for this test, which can be problematic efficiency-wise. For this reason, heuristics or special-purpose

Table 2.5: Example of an alternative attribute-value representation of molecules.

| Formula | H | He | Li | Be | B | C | N | O | F | ... |
|---|---|---|---|---|---|---|---|---|---|---|
| $H_2O$ | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | ... |
| $NO_2$ | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 0 | ... |
| $CO_2$ | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 2 | 0 | ... |

representations are employed. Next to the frequency measure, a lot of other measures of "interestingness" have been proposed in the context of subgraph mining algorithms, such as correlation, where one is interested in the graphs that correlate with the target value [Bringmann et al., 2006], or other constraints [Kramer et al., 2001; He and Singh, 2006].

### 2.3.3 Propositionalisation

In order to alleviate the efficiency problems of relational data mining techniques, propositionalisation techniques have been developed [Srinivasan and King, 1999; Flach and Lavrac, 2000; Kramer et al., 2001]. Propositionalisation consists of two steps. First, the relational representation is transformed into an attribute-value representation and then, efficient propositional learning algorithms can be applied. Propositionalisation approaches exist for several types of relational data [Lavrač et al., 1991], but here we concentrate on the propositionalisation of graphs in particular.

**Example 2.7** *One possibility to propositionalise a molecular graph into a fixed table is simply counting the number of elements in each molecule (Table 2.5).*

The approach in the above example may be easy to carry out, but turns out to be much too naive. For example, the graphs in Fig. 2.3b and Fig. 2.3c cannot be distinguished in this representation, while they have a different structure.

**Example 2.8** *Consider a set of patterns that were found by a hypothetical graph miner, shown in Fig. 2.5. For each of these patterns, it is checked whether it occurs in every graph. In this way, each graph is transformed into a vector (see Fig. 2.6).*

The advantage of propositionalisation is, once the data has been transformed into an attribute-value representation, many efficient propositional machine learning techniques can be used. Also techniques for feature selection and handling noise can be directly applied. The disadvantages are that it may take considerable time and effort, while there is loss of structural information and the possible exponential increase in the number of attributes [De Raedt, 1998]. It is also limited to a

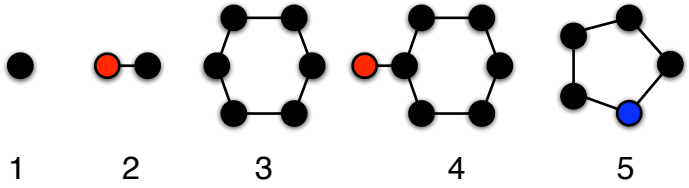


Figure 2.5: A set of 5 patterns found by a hypothetical graph miner.

restricted class of relational learning problems, since it cannot deal with recursive relations [Džeroski and Lavrač, 2001a].

## 2.4 Decision tree learning

Decision tree learning is one of the most popular machine learning methods. The reason for this is that decision trees combine a high predictive performance with a high interpretability. Moreover, there exist efficient algorithms to learn them.

Because we will use decision trees for the task of gene function prediction in the first part of the thesis, we will discuss the most important properties of decision tree learning in this section. First, we will explain how a decision tree is represented and how one can use it to make predictions on unseen data (Sect. 2.4.1), and then, we will show the various settings in which decision trees can be learned from data (Sect. 2.4.2).

### 2.4.1 Decision trees

A *decision tree* is a directed, rooted tree that represents a function from an input domain to a target $T$. Each internal node of a decision tree contains a *test* on an attribute and each leaf node contains a prediction about the target.

Figure 2.7 shows an example of a decision tree. This tree predicts whether a given molecule will be toxic, given tests on the properties that were given in Table 2.1. The prediction for an unseen example is the result of confronting the example to each test that is encountered in the tree. Starting at the root and depending on the outcome of each test, the appropriate edge is selected. Eventually, the example ends up in the leaf and the predicted value is obtained. A special case of a decision tree is a binary decision tree, which consists only of boolean tests. Figure 2.8 shows binary version of the decision tree shown in Fig. 2.7.

If a decision tree predicts a nominal class attribute, we call it a classification tree, if it predicts a numerical value, we call it a regression tree and if it predicts a probability, we call it a probability tree. While the tree in Fig. 2.7 has a target

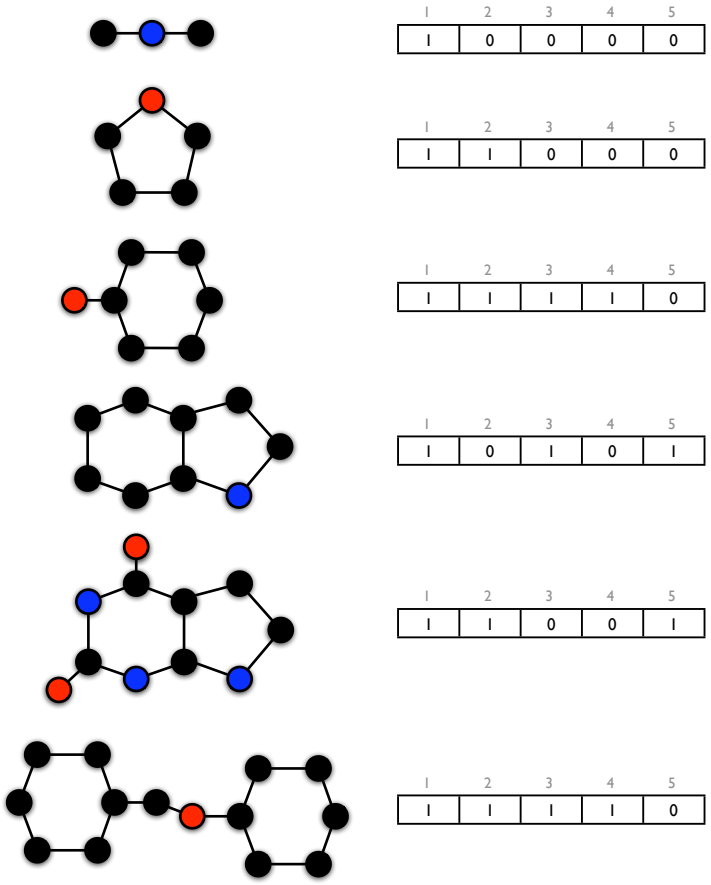


Figure 2.6: Propositionalisation of graphs into vectors.

attribute with binary values (the positive and negative class), this can also be extended to the setting in which multiple targets are predicted, which we will do in the first part of the thesis.

Decision trees have many advantages [Kramer and Widmer, 2001]. They obtain a good predictive performance, lend themselves to interpretation by domain experts and are robust to noise. Moreover, the learning of decision trees is efficient, as well as the procedure to make predictions from them.

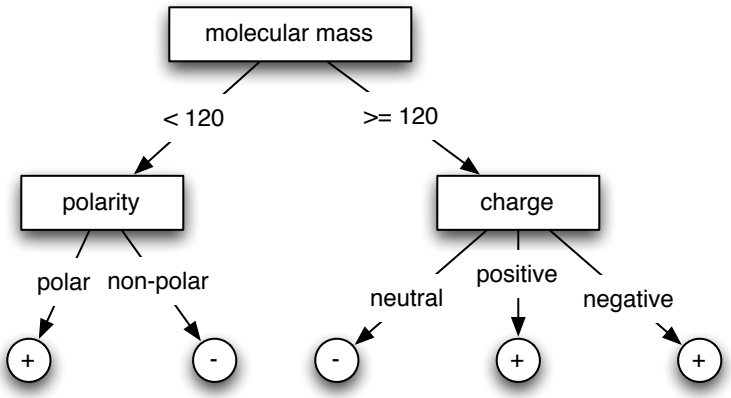


Figure 2.7: A classification tree predicting the toxicity of molecules. '+' means toxic, '-' means not toxic.

### 2.4.2 Top-down induction of decision trees

Decision tree learning refers to the task of learning a decision tree that is consistent with a given dataset. Because finding the smallest tree that exactly matches the data is an NP-hard problem [Zantema and Bodlaender, 2000], most algorithms that learn trees use a greedy search strategy. This does not guarantee that an optimal tree is found, but this approach works very well in practice.

Most algorithms for learning decision trees build trees top-down, from the roots towards the leaves. This is called *top-down induction of decision trees* (TDIDT) [Quinlan, 1986]. Various systems have implemented this approach, such as CART [Breiman et al., 1984], C4.5 [Quinlan, 1993] and J48, which is the reimplementation of C4.5 in the Weka data mining toolbox [Witten and Frank, 2005].

The way in which a TDIDT algorithm works is as follows (Algorithm 2.1). The algorithm starts with the root of the tree, trying to decide the best test to perform. The quality of a test is measured through its ability to split the examples in the node in homogeneous subsets w.r.t. the target attribute. Several heuristics have been proposed to this end, such as information gain [Quinlan, 1993], gain ratio [Quinlan, 1993] or Gini-index [Breiman et al., 1984]. Once a test has been chosen, it is put in the root and a child node of the root is created for each possible outcome of the test, creating subsets of examples based on their attribute values for that test. The algorithm proceeds with learning for each of these separate subsets a decision tree in exactly the same way: choose a test, divide the set into subsets and repeat the procedure on these subsets. As soon as a subset is found that is pure, meaning that all the examples in the subset have the same class, the procedure stops.

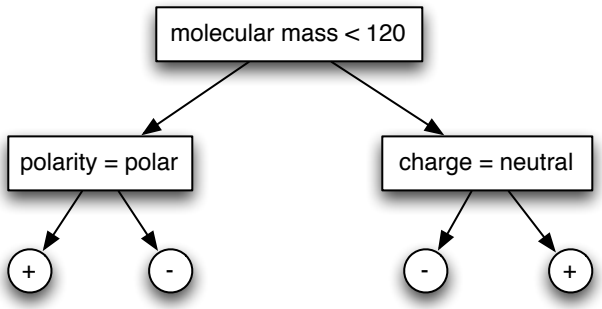


Figure 2.8: A binary decision tree for predicting molecular toxicity. All tests in the tree are boolean. Examples succeeding the test take the left branch, examples failing it the right branch.

**Algorithm 2.1** A generic TDIDT algorithm.

**function** TDIDT($E$: set of examples) **returns** decision tree

1: $\mathcal{T}$ := set of all possible tests
2: $\tau^*$ := $\arg\max_{\tau \in \mathcal{T}}$ quality$(\tau, E)$
3: **if** stop_criterion $(\tau^*, E)$
4: **return leaf**(local_model($E$))
5: **else**
6: $\mathcal{P}$ := partition induced on $E$ by $\tau^*$
7: **for all** $P_j$ in $P$
8: $T_j$ := TDIDT($P_j$)
9: **return node**$(\tau^*, \cup_j \{(j, T_j)\})$

In practice, there are several issues that are not covered by this simplified algorithm. For example, extensions have been proposed that specify how to perform tests on non-nominal attributes, how to use pruning criteria to stop growing the tree and how to predict probabilities or even multiple targets. More information about these extensions can be found in [Mitchell, 1997; Blockeel, 2007].

## 2.5 Measuring the quality of a learned model

An important aspect of machine learning is to assess the quality of the learned models. Predictive performance is an important measure, but one might also be interested in efficiency (e.g., the time to learn the model and to perform predictions) or interpretability (e.g., the possibility of acquiring insights from a knowledge dis-

Table 2.6: Contingency matrix for a binary classifier.

| | Real Positive | Real Negative |
|---|---|---|
| Predicted positive | TP | FP |
| Predicted negative | FN | TN |

covery point of view).

Here we focus on predictive performance measures. A lot of different measures exist and in a particular context it is not always straightforward to select an appropriate one. Therefore, it is important to understand what exactly an evaluation measure is measuring, such that, given the situation, the most sensible one can be selected.

When measuring the predictive performance of a model, ideally one is interested in the performance on the complete example space, which is unknown. In practice, only a limited amount of data is available. Usually a first part of this data, called the training data, is used to train the model and a second part, which has the same distribution as the first part and is called the test data, is used to estimate the performance. The training data cannot be used to measure performance, since it will return too optimistic estimates. However, when the amount of data is limited, one wants to use as much examples as possible to learn the model. A common way to solve this issue is known as crossvalidation. $N$-fold crossvalidation partitions the available data in $N$ equally sized folds. Then, the learning algorithm is repeated $N$ times, each time leaving out one fold as test set and using the remaining $N-1$ folds for training. The estimates computed on each test fold are then averaged to get a final estimate.

In this section, we will discuss several evaluation measures that will be used further on in the text. We only consider evaluation measures for classifiers, that is, models that predict a certain class. For numerical predictors, other evaluation measures are used. For more details, we refer to [Blockeel, 2007].

But first, we present the contingency matrix. Given a binary classification problem, and given the predictions of a binary classifier (predicting positive or negative), one can always make the matrix depicted in Table 2.6. Each example ends up in one of four different cells in the matrix. The true positives (TP) and the true negative (TN) are the examples that were correctly classified as positive and negative, respectively. The false positives (FP) are the negative examples that were incorrectly classified as positive. Finally, the false negatives (FN) are the positive examples that were incorrectly classified as negative. If $N$ is the number of examples in the dataset, then it is equal to $TP + FP + TN + FN$.

### 2.5.1 Accuracy

The most straightforward way to assess the predictive performance of a classifier is to count its number of correct predictions. This leads to the accuracy of a classifier: it corresponds to the fraction of correct predictions on a dataset. More precisely, given the contingency matrix, the accuracy of a model is given by

$$Acc = \frac{TP + TN}{N}$$

However, in some cases, accuracy is not a suitable evaluation measure. For example, when dealing with imbalanced class distributions, the accuracy may give a misleading view on a classifier's performance.

**Example 2.9** *Consider a dataset of HIV viruses of which 5% has developed resistance against the drug Indinavir. A classifier predicting every example to be non-resistant will obtain an accuracy of 95%, while it can be hardly considered a good model. On the other hand, a classifier that predicts 15% of the viruses to have developed resistance, of which 5% that really are resistant, then it has an accuracy of 90%, as it correctly classifies the 5% of resistant examples but misclassifies 10% of the non-resistant examples. So, based on accuracy, one would judge that the first classifier is better, though the second classifier is clearly more informative as it has learned relevant features to correctly classify viruses that have developed resistance.*

Also in the case where there are different misclassification costs, accuracy has some drawbacks. For example, the cost of failing to identify a rare illness in a patient (by incorrectly classifying him as healthy) could be very high, if for example it leads to the patient's death, when compared to the cost of incorrectly classifying a healthy patient, which would result in some unnecessary medical treatments.

### 2.5.2 ROC analysis

We start by introducing two new measures: the true positive rate (TPR) or the proportion of positive examples that is correctly classified and the false positive rate (FPR) or the proportion of negative examples that is correctly classified.

$$TPR = \frac{TP}{TP + FN}, \text{ and } FPR = \frac{FP}{FP + TN}$$

A cartesian coordinate system where the X and Y axes correspond to FPR and TPR respectively, is known as a Receiver Operation Characteristics (ROC) space. A binary classifier represents a single point in ROC space. The point (0,0) corresponds to a classifier that predicts all the examples as negative, while the point (1,1) corresponds to the classifier that always predicts positive. A classifier that randomly predicts examples as positive with probability $p$ and as negative

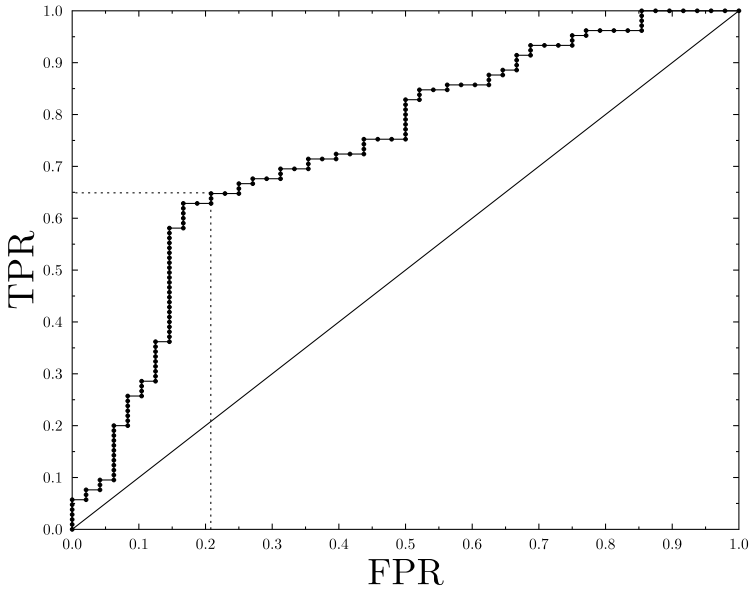


Figure 2.9: Example of a ROC curve.

with probability $1 - p$ is plotted as a point on the diagonal. A perfect classifier has a TPR of 1 and an FPR of 0.

When a classifier predicts probabilities, it is possible to construct a ROC curve by varying a classification threshold between 0 and 1. All examples with a predicted probability greater than the threshold value are classified as positive and the remaining as negative. A plot of a ROC curve is shown in Fig. 2.9.

The area under the ROC curve (AUROC) is a measure for how well the model can discriminate between positives and negatives. More precisely, it represents the probability that a positive and a negative example chosen randomly from the dataset are ordered correctly, that is, the negative example has a lower value than the positive example. Usually, AUROC is preferred over accuracy as it is not affected by imbalanced class distributions and it allows one to trade off the possibly different costs of incorrectly classifying a positive example as negative and vice versa. A perfect classifier has an AUROC of 1, a non-informative classifier an AUROC of 0.5.

Further background on ROC analysis in machine learning and data mining can be found in Provost and Fawcett [2001] and Fawcett [2006].

### 2.5.3 Precision and recall

Precision and recall are traditionally defined for a binary classification task with positive and negative classes. Precision is the proportion of positive predictions that are correct, and recall is the proportion of positive examples that are correctly predicted positive. That is,

$$Precision = \frac{TP}{TP + FP}, \quad \text{and} \quad Recall = \frac{TP}{TP + FN}.$$

**Example 2.10** *For the task of predicting resistance of HIV viruses, the precision is equal to the percentage of viruses that were predicted positive actually are positive and recall is the percentage of resistant viruses that were retrieved. For example, if 10% of the examples in the dataset is resistant, then a classifier that predicts 8% of these examples correctly, while also predicting 12% of the other examples incorrectly as positive, has a precision of 40% and a recall of 80%.*

As for TPR and FPR, depending on the learning task, there is usually a trade-off to be considered between precision and recall.

## 2.6 Summary

In this chapter, we have explained the basic concepts from machine learning and data mining that will be built upon in the rest of this thesis. We now summarise the main concepts introduced in the chapter and their relevance to the rest of this thesis.

- We have introduced several ways to represent structured data and explained the advantages and shortcomings of attribute-value learning and relational learning. Relational learning techniques are much more expressive than their propositional counterparts, but this comes at the price of efficiency.
- We have used the representation of molecules as an illustrating example. Since we will focus on the task of structure-activity learning in the second part of the thesis, we have focused in particular on graph mining techniques.
- We have explained the concept of decision trees. In the first part of the thesis, we will extend the decision tree learning framework to be able to handle the task of gene function prediction.
- We have discussed several predictive performance measures that will be used in the rest of this thesis.

# Part I

# Structured Output Learning for the Prediction of Gene Function

# Outline Part I

In the first part of the thesis, we focus on the learning task of hierarchical multi-label classification (HMC). HMC differs from normal classification in two ways: (1) a single example may belong to multiple classes simultaneously; and (2) the classes are organised in a hierarchy: an example that belongs to some class automatically belongs to all its superclasses (we call this the *hierarchy constraint*). The HMC task can be interpreted as a structured output learning problem, since the goal is to predict a set of paths in the hierarchy that are consistent with the hierarchy constraint, that is, for each predicted class, all classes on the path from that class to the root of the hierarchy should be predicted as well.

Examples of HMC problems are found in several domains, including text classification [Rousu et al., 2006], functional genomics [Barutcuoglu et al., 2006], and object recognition [Stenger et al., 2007]. Throughout the next two chapters, we will focus on the application of functional genomics, where the task is to predict the functions of genes. Biologists have a set of possible functions that genes may have, and these functions are organised in a hierarchies, such as MIPS's FunCat (structured as a tree) or the Gene Ontology (structured as a directed acyclic graph). It is known that a single gene may have multiple functions. Since the completion of several genome projects, which have generated the complete genome sequence of many organisms, identifying genes in the sequences and assigning biological functions to them has now become a key challenge in modern biology. This last step is often guided by automatic discovery processes which interact with the laboratory experiments.

In order to understand the interactions between different genes, it is important to obtain an interpretable model. The motivation for using decision trees in this context is the following: they are a well-known type of classifiers that can be learned efficiently from large datasets, produce accurate predictions and can lead to knowledge that provides insight in the biology behind the predictions, as demonstrated by Clare and King [2003].

We will address HMC and its application to functional genomics from two different angles. In Chapter 3, we will investigate HMC from a machine learning point of view. We will introduce three different learning approaches that solve the

HMC task, instantiate them in decision tree learners and compare the decision trees in terms of predictive performance, efficiency and model size. For the evaluation, we use datasets from functional genomics. In Chapter 4, we approach HMC from a biological perspective: we focus on the application domain of gene function prediction and compare our HMC approach to several other methods in the biomedical literature in terms of predictive performance, efficiency, interpretability and usability.

# Chapter 3

# Decision Trees for Hierarchical Multi-label Classification

## 3.1 Introduction

In this chapter, we will present several approaches to the induction of decision trees for HMC, as well as an extensive experimental study on several functional genomics datasets. As mentioned in the outline, a single gene may have multiple functions and biologists have organised these functions in a hierarchy (see Fig. 3.1 for an example of the FunCat classification scheme). In order to understand the interactions between different genes, it is important to obtain an interpretable model, which explains our choice for decision trees.

A first approach transforms an HMC task into a separate binary classification task for each class in the hierarchy and applies an existing classification algorithm. We refer to it as the SC (single-label classification) approach. This technique has several disadvantages. First, it is *inefficient*, because the learner has to be run $|C|$ times, with $|C|$ the number of classes, which can be hundreds or thousands in some applications. Second, it often results in *learning from strongly skewed class distributions*: in typical HMC applications, classes at lower levels of the hierarchy often have very small frequencies, while the frequency of classes at higher levels tends to be very high. This is due to the *hierarchy constraint*, which postulates that an example belonging to some class automatically belongs to all its superclasses. Many learners have problems with strongly skewed class distributions [Weiss and Provost, 2003]. Third, from the knowledge discovery point of view, the learned models *identify features relevant for one class*, rather than identifying features

```
1 METABOLISM
1.1 amino acid metabolism
1.1.3 assimilation of ammonia, metabolism of the glutamate group
1.1.3.1 metabolism of glutamine
1.1.3.1.1 biosynthesis of glutamine
1.1.3.1.2 degradation of glutamine
...
1.2 nitrogen, sulfur, and selenium metabolism
...
2 ENERGY
2.1 glycolysis and gluconeogenesis
...
```

Figure 3.1: A small part of the hierarchical FunCat classification scheme [Mewes et al., 1999].

with high overall relevance. Finally, *the hierarchy constraint is not taken into account*, i.e. it is not automatically imposed that an instance belonging to a class should belong to all its superclasses.

A second approach is to adapt the SC method, so that this last issue is dealt with. Some authors have proposed to hierarchically combine the class-wise models in the prediction stage, so that a classifier constructed for a class $c$ can only predict positive if the classifier for the parent class of $c$ has predicted positive [Barutcuoglu et al., 2006; Cesa-Bianchi et al., 2006]. In addition, one can also take the hierarchy constraint into account during training by restricting the training set for the classifier for class $c$ to those instances belonging to the parent class of $c$ [Cesa-Bianchi et al., 2006]. This approach is called the HSC (hierarchical single-label classification) approach throughout the chapter.

A third approach is to develop learners that learn a single multi-label model that predicts all the classes of an example at once [Clare, 2003; Blockeel et al., 2006]. Next to taking the hierarchy constraint into account, this approach is also able to identify features that are relevant to all classes. We call this the HMC approach.

The contributions of this chapter are threefold:

- We introduce three decision tree approaches towards HMC tasks: (1) learning a separate binary decision tree for each class label (SC), (2) learning and applying such single-label decision trees in a hierarchical way (HSC), and (3) learning one tree that predicts all classes at once (HMC). The HSC approach has not been considered before in the context of decision trees.

- We compare the approaches by performing an extensive experimental evaluation in terms of predictive performance, efficiency and model size on 24 datasets from yeast functional genomics, using as classification schemes MIPS's FunCat [Mewes et al., 1999] (tree structure) and the Gene Ontology [Ashburner et al., 2000] (DAG structure). The latter results in datasets with (on average) 4000 class labels, which underlines the scalability of the

approaches to large class hierarchies.

- When dealing with the highly skewed class distributions that are characteristic for the HMC setting, precision-recall curves are the most suitable evaluation tool [Davis and Goadrich, 2006]. We propose several ways to perform a precision-recall based analysis in domains with multiple (hierarchically organised) class labels and discuss the difference in their behavior.

The contents of this chapter are the result of joint work with Celine Vens, Jan Struyf and Hendrik Blockeel at the Katholieke Universiteit Leuven and Sašo Džeroski at the Jožef Stefan Institute Ljubljana. They have been published in [Blockeel et al., 2006; Vens et al., 2008].

The chapter is organised as follows. We start by discussing previous work in Sect. 3.2. Section 3.3 presents the three decision tree methods for HMC in detail. In Sect. 3.4, we describe how to extend the algorithms towards DAG structured class hierarchies. In Sect. 3.5, we propose the precision-recall based performance measures, used for the empirical study described in Sect. 3.6. Finally, we conclude in Sect. 3.7.

## 3.2 Related work

In this chapter, we will give a general overview of related work, while research that is specific to the area of functional genomics will be discussed in detail in the next chapter.

Much work in hierarchical multi-label classification (HMC) has been motivated by text classification. Rousu et al. [2006] present the state of the art in this domain, which consists mostly of Bayesian and kernel-based classifiers.

Koller and Sahami [1997] consider a hierarchical text classification problem setting where each text document belongs to exactly one class at the bottom level of a topic hierarchy. For each topic in an internal node of the hierarchy, a Bayesian classifier is learned that distinguishes between the possible subtopics, using only those training instances that belong to the parent topic. Test documents are then classified by filtering them through the hierarchy, predicting one topic at each level, until the documents reach the bottom level, thereby ensuring the hierarchy constraint. Errors made at higher levels of the hierarchy are unrecoverable at the lower levels. The procedure is similar to the HSC approach. Nevertheless, as only one path in the hierarchy is predicted, the method is not strictly multi-label. Another difference with HSC is that the node classifiers are not binary.

In the work of Cesa-Bianchi et al. [2006], every data instance is labeled with a set of class labels, which may belong to more than one path in the hierarchy. Instances can also be tagged with labels belonging to a path that does not end in a leaf. At each node of the (tree-shaped) taxonomy a binary linear threshold classifier is built, using as training instances only those instances belonging to

the node's parent class. This is thus an HSC method. The parameters of the classifier are trained incrementally: at each timestamp, an example is presented to the current set of classifiers, the predicted labels are compared to the real labels, and the classifiers' parameters are updated. In that process, a classifier can only predict positive if its parent classifier has predicted positive, ensuring that the hierarchy constraint is satisfied.

Barutcuoglu et al. [2006] presented a two-step approach where support vector machines (SVMs) are learned for each class separately, and then combined using a Bayesian network model so that the predictions are consistent with the hierarchy constraint.

Rousu et al. [2006] presented a more direct approach that does not require a second step to make sure that the hierarchy constraint is satisfied. Their approach is based on a large margin method for structured output prediction [Taskar et al., 2003; Tsochantaridis et al., 2005]. Such work defines a joint feature map $\Psi(x, y)$ over the input space $X$ and the output space $Y$. In the context of HMC, the output space $Y$ is the set of all possible subtrees of the class hierarchy. Next, it applies SVM based techniques to learn the weights $w$ of the discriminant function $F(x, y) = \langle w, \Psi(x, y) \rangle$, with $\langle \cdot, \cdot \rangle$ the dot product. The discriminant function is then used to classify a (new) instance $x$ as $\text{argmax}_{y \in Y} F(x, y)$. There are two main challenges that must be tackled when applying this approach to a structured output prediction problem: (a) defining $\Psi$, and (b) finding an efficient way to compute the argmax function (the range of this function is $Y$, which is of size exponential in the number of classes). Rousu et al. [2006] describe a suitable $\Psi$ and propose an efficient method based on dynamic programming to compute the argmax.

From the point of view of knowledge discovery, it is sometimes useful to obtain more interpretable models, such as decision trees, which is the kind of approach we study here.

Clare and King [2001] presented a decision tree method for multi-label classification in the context of functional genomics. In their approach, a tree predicts not a single class but a vector of boolean class variables. They propose a simple adaptation of C4.5 to learn such trees: where C4.5 normally uses class entropy for choosing the best split, their version uses the sum of entropies of the class variables. Clare [2003] extended the method to predict classes on several levels of the hierarchy, assigning a larger cost to misclassifications higher up in the hierarchy, and presented an evaluation on the twelve datasets that we also use here.

Blockeel et al. [1998, 2002] proposed the idea of using predictive clustering trees for HMC tasks. The research in this chapter is the result of elaborating on that idea.

Geurts et al. [2006] presented a decision tree based approach related to predictive clustering trees. They start from a different definition of variance and then kernelise this variance function. The result is a decision tree induction system

that can be applied to structured output prediction using a method similar to the large margin methods mentioned above [Tsochantaridis et al., 2005; Taskar et al., 2003].

## 3.3 Decision tree approaches for HMC

We start this section by defining the HMC task more formally (Sect. 3.3.1). Next, we present the framework of predictive clustering trees (Sect. 3.3.2), which will be used to instantiate three decision tree algorithms for HMC tasks: an HMC algorithm (Sect. 3.3.3), an SC algorithm (Sect. 3.3.4), and an HSC algorithm (Sect. 3.3.5). Section 3.3.6 compares the three algorithms at a conceptual level. In this section, we assume that the class hierarchy has a tree structure. Section 3.4 will discuss extensions towards hierarchies structured as a DAG.

### 3.3.1 Formal task description

We define the task of hierarchical multi-label classification as follows:

**Given:**

- an instance space $X$,
- a class hierarchy $(C, \leq_h)$, where $C$ is a set of classes and $\leq_h$ is a partial order (structured as a rooted tree for now) representing the superclass relationship (for all $c_1, c_2 \in C$: $c_1 \leq_h c_2$ if and only if $c_1$ is a superclass of $c_2$),
- a set $T$ of examples $(x_i, S_i)$ with $x_i \in X$ and $S_i \subseteq C$ such that $c \in S_i \Rightarrow \forall c' \leq_h c : c' \in S_i$, and
- a quality criterion $q$ (which typically rewards models with high predictive accuracy and low complexity).

**Find:** a function $f : X \rightarrow 2^C$ (where $2^C$ is the power set of $C$) such that $f$ maximises $q$ and $c \in f(x) \Rightarrow \forall c' \leq_h c : c' \in f(x)$. We call this last condition the hierarchy constraint.

In this chapter, the function $f$ is represented with decision trees.

### 3.3.2 Predictive clustering trees

The decision tree methods that we present in the next sections are set in the predictive clustering tree (PCT) framework [Blockeel et al., 1998]. This framework views a decision tree as a hierarchy of clusters: the top-node corresponds to one cluster containing all data, which is recursively partitioned into smaller clusters while moving down the tree. PCTs are constructed so that each split maximally

**Algorithm 3.1** The top-down induction algorithm for PCTs. $I$ denotes the current training instances, $t$ an attribute-value test, $\mathcal{P}$ the partition induced by $t$ on $I$, and $h$ the heuristic value of $t$. The superscript $^*$ indicates the current best test and its corresponding partition and heuristic. The functions Var, Prototype, and Acceptable are described in the text.

**procedure** PCT($I$) **returns** tree
1: $(t^*, \mathcal{P}^*) = \text{BestTest}(I)$
2: **if** $t^* \neq none$
3: **for each** $I_k \in \mathcal{P}^*$
4: $tree_k = \text{PCT}(I_k)$
5: **return** node($t^*$, $\bigcup_k \{tree_k\}$)
6: **else**
7: **return** leaf(Prototype($I$))

**procedure** BestTest($I$)
1: $(t^*, h^*, \mathcal{P}^*) = (none, 0, \emptyset)$
2: **for each** possible test $t$
3: $\mathcal{P}$ = partition induced by $t$ on $I$
4: $h = \text{Var}(I) - \sum_{I_k \in \mathcal{P}} \frac{|I_k|}{|I|} \text{Var}(I_k)$
5: **if** $(h > h^*) \wedge \text{Acceptable}(t, \mathcal{P})$
6: $(t^*, h^*, \mathcal{P}^*) = (t, h, \mathcal{P})$
7: **return** $(t^*, \mathcal{P}^*)$

reduces intra-cluster variance. They can be applied to both clustering and prediction tasks, and have clustering trees and (multi-objective) classification and regression trees as special cases.

PCTs [Blockeel et al., 1998] can be constructed with a standard "top-down induction of decision trees" (TDIDT) algorithm, similar to CART [Breiman et al., 1984] or C4.5 [Quinlan, 1993]. The algorithm (Table 3.1) takes as input a set of training instances $I$. The main loop (Table 3.1, BestTest) searches for the best acceptable attribute-value test that can be put in a node. If such a test $t^*$ can be found then the algorithm creates a new internal node labeled $t^*$ and calls itself recursively to construct a subtree for each subset (cluster) in the partition $\mathcal{P}^*$ induced by $t^*$ on the training instances. To select the best test, the algorithm scores the tests by the reduction in variance (which is to be defined further) they induce on the instances (Line 4 of BestTest). Maximising variance reduction maximises cluster homogeneity and improves predictive performance. If no acceptable test can be found, that is, if no test significantly reduces variance, then the algorithm creates a leaf and labels it with a representative case, or prototype, of the given instances.

The above description is not very different from that of standard decision tree learners. The main difference is that PCTs treat the variance and prototype functions as parameters, and these parameters are instantiated based on the learning task at hand. To construct a regression tree, for example, the variance function returns the variance of the given instances' target values, and the prototype is the average of their target values. By appropriately defining the variance and prototype functions, PCTs have been used for clustering [Blockeel et al., 1998; Struyf and Džeroski, 2007], multi-objective classification and regression [Blockeel et al., 1998, 1999; Struyf and Džeroski, 2006; Demšar et al., 2006], and time series data

analysis [Džeroski et al., 2006]. Section 3.3.3 shows how PCTs can, in a similar way, be applied to hierarchical multi-label classification.

The PCT framework is implemented in the Inductive Logic Programming system TILDE [Blockeel et al., 1998] and in the CLUS system. We will use the CLUS implementation. More information about CLUS can be found at `http://www.cs.kuleuven.be/~dtai/clus`.

### 3.3.3 Clus-HMC: An HMC decision tree learner

To apply PCTs to the task of hierarchical multi-label classification, the variance and prototype parameters are instantiated as follows.

First, the example labels are represented as vectors with boolean components; the $i$'th component of the vector is 1 if the example belongs to class $c_i$ and 0 otherwise. It is easily checked that the arithmetic mean of a set of such vectors contains as $i$'th component the proportion of examples of the set belonging to class $c_i$. We define the variance of a set of examples as the average squared distance between each example's label $v_k$ and the set's mean label $\overline{v}$, i.e.,

$$Var(S) = \frac{\sum_k d(v_k, \overline{v})^2}{|S|}.$$

In the HMC context, it makes sense to consider similarity on higher levels of the hierarchy more important than similarity on lower levels. To that aim, we use a weighted Euclidean distance

$$d(v_1, v_2) = \sqrt{\sum_i w(c_i) \cdot (v_{1,i} - v_{2,i})^2},$$

where $v_{k,i}$ is the $i$'th component of the class vector $v_k$ of an instance $x_k$, and the class weights $w(c)$ decrease with the depth of the class in the hierarchy (e.g., $w(c) = w_0^{\text{depth}(c)}$, with $0 < w_0 < 1$). Consider for example the class hierarchy shown in Fig. 3.2, and two examples $(x_1, S_1)$ and $(x_2, S_2)$ with $S_1 = \{1, 2, 2.2\}$ and $S_2 = \{2\}$. Using a vector representation with consecutive components representing membership of class 1, 2, 2.1, 2.2 and 3, in that order,

$$d([1, 1, 0, 1, 0], [0, 1, 0, 0, 0]) = \sqrt{w_0 + w_0^2}.$$

The heuristic for choosing the best test for a node of the tree is then maximisation of the variance reduction as discussed in Sect. 3.3.2, with the above definition of variance. Note that this essentially corresponds to converting the example labels to 0/1 vectors and then using the same variance definition as is used when applying PCTs to multi-objective regression [Blockeel et al., 1998, 1999], but with appropriate weights. In the single-label case, this heuristic is in turn identical to

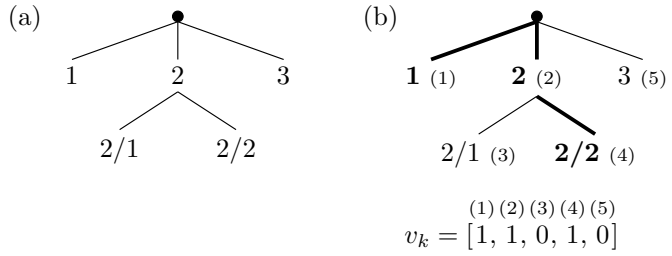


Figure 3.2: (a) A small hierarchy. Class label names reflect the position in the hierarchy, e.g., '2/1' is a subclass of '2'. (b) The set of classes {1,2,2.2}, indicated in bold in the hierarchy, and represented as the vector $v_k$.

the heuristic used in regression tree learners such as CART [Breiman et al., 1984], and equivalent to the Gini index used by CART in classification tree mode.

A classification tree stores in a leaf the majority class for that leaf; this class will be the tree's prediction for examples arriving in the leaf. But in our case, since an example may have multiple classes, the notion of "majority class" does not apply in a straightforward manner. Instead, the mean $\bar{v}$ of the vectors of the examples in that leaf is stored; in other words, the prototype function returns $\bar{v}$. Fig. 3.3a shows a simple HMC tree for the hierarchy of Fig. 3.2.

Recall that $\bar{v}_i$ is the proportion of examples in the leaf belonging to class $c_i$, which can be interpreted as the probability that an example arriving in the leaf has class $c_i$. If $\bar{v}_i$ is above some threshold $t_i$, the example is predicted to belong to class $c_i$. To ensure that the predictions fulfil the hierarchy constraint (whenever a class is predicted its superclasses are also predicted), it suffices to choose $t_i \leq t_j$ whenever $c_i \leq_h c_j$.

Exactly how the thresholds should be chosen is a question that we do not address here. Depending on the context, a user may want to set the thresholds such that the resulting classifier has maximal predictive accuracy, high precision at the cost of lower recall or vice versa, maximal F1-score (which reflects a particular trade-off between precision and recall), minimal expected misclassification cost (where different types of mistakes may be assigned different costs), maximal interpretability or plausibility of the resulting model, etc. Instead of committing to a particular rule for choosing the threshold, we will study the performance of the predictive models using threshold-independent measures. More precisely, we will use precision-recall curves (as will be clear in Sect. 3.5).

Finally, the function Acceptable in Table 3.1 verifies for a given test that the number of instances in each subset of the corresponding partition $\mathcal{P}$ is at least *mincases* (a parameter) and that the variance reduction is significant according to a statistical $F$-test. We call the resulting algorithm CLUS-HMC.

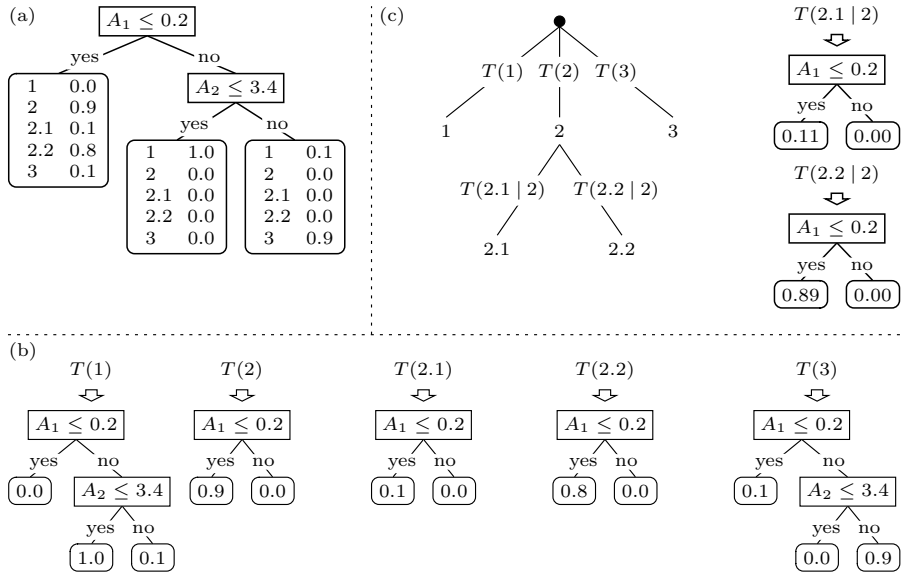


Figure 3.3: (a) HMC: one tree predicting, in each leaf, the probability for each class in the hierarchy. (b) SC: a separate tree $T(c_i)$ for each class $c_i$. (c) HSC: a separate tree for each hierarchy edge. The left part of (c) shows how the HSC trees are organised in the class hierarchy. The right part shows $T(2.1 \mid 2)$ and $T(2.2 \mid 2)$; trees $T(1)$, $T(2)$, and $T(3)$ are identical to those of SC. Note that the leaves of $T(2.1 \mid 2)$ and $T(2.2 \mid 2)$ predict conditional probabilities.

### 3.3.4 Clus-SC: Learning a separate tree for each class

The second approach that we consider builds a separate tree for each class in the hierarchy (Fig. 3.3b). Each of these trees is a single-label binary classification tree. Assume that the tree learner takes as input a set of examples labeled positive or negative. To construct the tree for class $c$ with such a learner, we label the class $c$ examples positive and all the other examples negative. The resulting tree predicts the probability that a new instance belongs to $c$. We refer to this method as single-label classification (SC).

In order to classify a new instance, SC thresholds the predictions of the different single-label trees, similar to Clus-HMC. Note, however, that this does not guarantee that the hierarchy constraint holds, even if the thresholds are chosen such that $t_i \leq t_j$ whenever $c_i \leq_h c_j$. Indeed, the structure of the SC trees can be different from that of their parent class's SC tree[1], and therefore, the tree built for,

[1]Fig. 3.3 was chosen to show that the different approaches (HMC/SC/HSC) are able to express the same concept; the SC trees all have the same structure and are subtrees of the Clus-HMC

e.g., class 2.1 may very well predict a higher probability than the tree built for class 2 for a given instance. In practice, post-processing techniques can be applied to ensure that a class probability does not exceed its parent class probability. This problem does not occur with Clus-HMC; Clus-HMC always predicts smaller probabilities for specific classes than for more general classes.

The class-wise trees can be constructed with any classification tree induction algorithm. Note that Clus-HMC reduces to a single-label binary classification tree learner when applied to such data; its class vector then reduces to a single component and its heuristic reduces to Cart's Gini index [Breiman et al., 1984], as pointed out in Sect. 3.3.3. We can therefore use the same induction algorithm (Clus-HMC) for both the HMC and SC approaches. This makes the results easier to interpret. To check the validity of our particular implementation, we also ran the binary decision tree learner M5' from Weka [Witten and Frank, 2005] on the same datasets. It turns out that on these binary classification tasks, Clus-HMC performed even slightly better than M5'. We call the SC approach with Clus-HMC as decision tree learner Clus-SC.

### 3.3.5 Clus-HSC: Learning a separate tree for each hierarchy edge

Building a separate decision tree for each class has several disadvantages, such as the possibility of violating the hierarchy constraint. In order to deal with this issue, the Clus-SC algorithm can be adapted as follows (Fig. 3.3c).

For a non top-level class $c$, it holds that an instance can only belong to $c$ if it belongs to $c$'s parent $par(c)$. An alternative approach to learning a tree that directly predicts $c$, is therefore to learn a tree that predicts $c$ given that the instance belongs to $par(c)$. Learning such a tree requires fewer training instances: only the instances belonging to $par(c)$ are relevant. The subset of these instances that also belong to $c$ become the positive instances and the other instances (those belonging to $par(c)$ but not to $c$) the negative instances. The resulting tree predicts the conditional probability $P(c|par(c))$. W.r.t. the top-level classes, the approach is identical to Clus-SC, i.e., all training instances are used.

To make predictions for a new instance, we use the product rule $P(c) = P(c \mid par(c)) \cdot P(par(c))$ (for non top-level classes). This rule applies the trees recursively, starting from the tree for a top-level class. For example, to compute the probability that the instance belongs to class 2.2, we first use the tree for class 2 to predict $P(2)$ and next the tree for class 2.2 to predict $P(2.2 \mid 2)$. The resulting probability is then $P(2.2) = P(2.2 \mid 2) \cdot P(2)$. Again, these probabilities are thresholded to obtain the predicted set of classes. As with Clus-HMC, to ensure that this set fulfils the hierarchy constraint, it suffices to choose a threshold

tree. In general, this is not the case.

Table 3.1: Comparing the three decision tree approaches at a conceptual level.

| | CLUS-HMC | CLUS-HSC | CLUS-SC |
|---|---|---|---|
| efficiency | + | + | - |
| dealing with imbalanced class distr. | ? | +/- | - |
| obeying the hierarchy constraint | + | + | - |
| identifying global features | + | - | - |

$t_i \leq t_j$ whenever $c_i \leq_h c_j$. We call the resulting algorithm CLUS-HSC (hierarchical single-label classification).

### 3.3.6 Comparison between Clus-HMC, Clus-SC and Clus-HSC

To conclude this section, we compare the three proposed approaches (HMC, SC and HSC) at a conceptual level, according to the properties mentioned in the introduction: the efficiency of learning the models, how skewed class distributions are dealt with, whether the hierarchy constraint is obeyed, and whether global or local features are identified. Other comparison measures, such as predictive performance and model size, will be investigated in depth in the experiments section. Table 3.1 gives an overview.

Concerning efficiency, CLUS-HMC is expected to be more efficient than CLUS-SC: although building one HMC tree will likely take longer than building one SC tree, CLUS-SC still needs to build $|C|$ of those trees, with $|C|$ the number of classes in the hierarchy. The CLUS-HSC algorithm is expected to be more efficient than CLUS-SC, since smaller training sets are used for constructing the trees. Experimental evaluation will have to demonstrate how CLUS-HMC, CLUS-SC and CLUS-HSC relate in practice.

As mentioned in the introduction, CLUS-SC has to deal with highly skewed class distributions for many of the trees it builds. CLUS-HSC reduces the training data for each class by discarding negative examples that do not belong to the parent class. In most cases, this yields a more balanced class distribution, although there is a small probability that the distribution becomes even more skewed.[2] On average we expect CLUS-HSC to suffer less from imbalanced class distributions

[2]Suppose we have 200 examples, of which 100 belong to class 1 and 20 to class 1.1; then when learning class 1.1 from the whole set, 10% of the training examples are positive, while when learning from examples of class 1 only, 20% are positive. So, the problem becomes better balanced. If, on the other hand, among the 100 class 1 examples, 90 belong to 1.1, then the original distribution has 90/200 = 45% positives, whereas when learning from class 1 examples only we have 90% positives: a more skewed dataset. Generally, the problem will become more balanced for classes $c$ where $\frac{N_c}{N} + \frac{N_c}{N_{par(c)}} < 1$ ($N$ denotes the number of examples and $par(c)$ the parent class of $c$).

than Clus-SC. For Clus-HMC, which learns all classes at once, it is difficult to estimate the effect of individual imbalanced class distributions.

As explained before, both Clus-HMC and Clus-HSC obey the hierarchy constraint if appropriate threshold values are chosen for each class (e.g., if all thresholds are the same), while Clus-SC does not.

Finally, whereas the models found by Clus-HSC and Clus-SC will contain features relevant for predicting one particular class, Clus-HMC will identify features with high overall relevance.

## 3.4 Hierarchies structured as DAGs

Until now, we have assumed that the class hierarchy is structured as a rooted tree. In this section, we discuss the issues that arise when dealing with more general hierarchies that are structured as directed acyclic graphs (DAGs). Such a class structure occurs when a given class can have more than one parent class in the hierarchy. An example of such a hierarchy is the Gene Ontology [Ashburner et al., 2000], a biological classification hierarchy for genes. In general, a classification scheme structured as a DAG can have two interpretations: if an instance belongs to a class $c$, then it either belongs also to all superclasses of $c$, or it belongs also to at least one superclass of $c$. We focus on the first case, which corresponds to the "multiple inheritance" interpretation, where a given class inherits the properties (classes) of all its parents. This interpretation is correct for the Gene Ontology.

In the following sections, we discuss the issues that arise when dealing with a DAG type class hierarchy, and discuss the modifications that are required to the algorithms discussed in the previous section to be able to deal with such hierarchies. Obviously, Clus-SC requires no changes because this method ignores the hierarchical structure of the classes.

### 3.4.1 Adaptations to Clus-HMC

Clus-HMC computes the variance based on the weighted Euclidean distance between class vectors, where a class $c$'s weight $w(c)$ depends on the depth of $c$ in the class hierarchy (e.g., $w(c) = w_0^{\text{depth}(c)}$). When the classes are structured as a DAG, however, the depth of a class is no longer unique: a class may have several depths, depending on the path followed from a top-level class to the given class (see for instance class $c_6$ in Fig. 3.4a). As a result, the class weights are no longer properly defined. We therefore propose the following approach. Observe that $w(c) = w_0^{\text{depth}(c)}$ can be rewritten as the recurrence relation $w(c) = w_0 \cdot w(par(c))$, with $par(c)$ the parent class of $c$, and the weights of the top-level classes equal to $w_0$. This recurrence relation naturally generalises to hierarchies where classes may have multiple parents by replacing $w(par(c))$ by an aggregation function computed

over the weights of $c$'s parents. Depending on the aggregation function used (sum, min, max, average), we obtain the following approaches:

- $w(c) = w_0 \sum_j w(par_j(c))$ is equivalent to flattening the DAG into a tree (by copying the subtrees that have multiple parents) and then using $w(c) = w_0^{\text{depth}(c)}$. The more paths in the DAG lead to a class, the more important this class is considered by this method. A drawback is that there is no guarantee that $w(c) < w(par_j(c))$. For example, in Fig. 3.4a, the weight of class $c_6$ is larger than the weights of both its parents.

- $w(c) = w_0 \cdot \min_j w(par_j(c))$ has the advantage that it guarantees $\forall c, j : w(c) < w(par_j(c))$. A drawback is that it assigns a small weight to a class that has multiple parents and that appears both close to the top-level and deep in the hierarchy.

- $w(c) = w_0 \cdot \max_j w(par_j(c))$ guarantees a high weight for classes that appear close to the top-level of the hierarchy. It does not satisfy $w(c) < w(par_j(c))$, but still yields smaller weights than $w(c) = w_0 \sum_j w(par_j(c))$.

- $w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$ can be considered a compromise in between the "min" and "max" approaches.

We compare the above weighting schemes in the experimental evaluation. Note that all the weighting schemes become equivalent for tree shaped hierarchies.

### 3.4.2 Adaptations to Clus-HSC

Recall that Clus-HSC builds models that predict $P(c \mid par(c))$. This approach can be trivially extended to DAG structured hierarchies by creating one model for each combination of a parent class with one of its children (or equivalently, one model for each hierarchy edge) predicting $P(c \mid par_j(c))$ (Fig. 3.4b). To make a prediction, the product rule $P(c) = P(c \mid par_j(c)) \cdot P(par_j(c))$ can be applied for each parent class $par_j(c)$, and will yield a valid estimate of $P(c)$ based on that parent. In order to obtain an estimate of $P(c)$ based on all parent classes, we aggregate over the parent-wise estimates.

Recall that Clus-HSC fulfills the hierarchy constraint in the context of tree structured class hierarchies. We want to preserve this property in the case of DAGs. To that aim, we use as aggregate function the minimum of the parent-wise estimates, i.e., $P(c) = \min_j P(c|par_j(c)) \cdot P(par_j(c))$. Clus-HSC applies this rule in a top-down fashion (starting with the top-level classes) to compute predicted probabilities for all classes in the hierarchy. Fig. 3.4b illustrates this process.

Instead of building one tree for each hierarchy edge, one could consider building a tree for each hierarchy node and using as training set for such a tree the instances labeled with all parent classes. This would yield trees predicting $P(c \mid \bigwedge par_j(c))$.

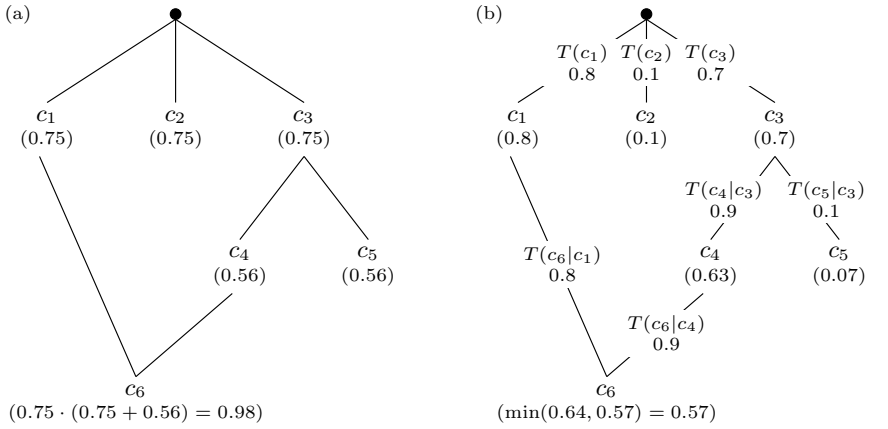


Figure 3.4: (a) A class hierarchy structured as a DAG. The class-wise weights computed for Clus-HMC with the weighting scheme $w(c) = w_0 \sum_j w(par_j(c))$ and $w_0 = 0.75$ are indicated below each class. (b) The trees constructed by Clus-HSC. Assume that these trees predict, for a given test instance, the conditional probabilities indicated below each tree. Clus-HSC then predicts the probability of a given class $c$ with the combining rule $P(c) = \min_j P(c \mid par_j(c)) \cdot P(par_j(c))$ (indicated below each class).

While this approach builds fewer trees, it has two important disadvantages. First, the number of training instances per tree can become very small (only the instances that belong to all parent classes are used). Second, the predicted class probabilities are now given by the rule $P(c) = P(c \mid \bigwedge par_j(c)) \cdot P(\bigwedge par_j(c))$, and it is unclear how the last term of this rule can be estimated for a test example. Clus-HSC therefore relies on the approach outlined above with one model per hierarchy edge.

## 3.5 Predictive performance measures for HMC

After having proposed three decision tree methods for HMC tasks with DAG structured class hierarchies, our next step is to compare their predictive performance, model size, and induction times. Before proceeding, we discuss how to evaluate the predictive performance of the classifiers.

### 3.5.1 Hierarchical loss

Cesa-Bianchi et al. [2006] have defined a hierarchical loss function that considers mistakes made at higher levels in the class hierarchy more important than mistakes

made at lower levels. The hierarchical loss function for an instance $x_k$ is defined as follows:

$$l_H(x_k) = \sum_i [v_{k,i} \neq y_{k,i} \; and \; \forall c_j \leq_h c_i : v_{k,j} = y_{k,j}],$$

where $i$ iterates over all class labels, $v$ represents the predicted class vector, and $y$ the real class vector. In the work of Cesa-Bianchi et al., the class hierarchy is structured as a tree, and thus, it penalises the first mistake along the path from the root to a node. In the case of a DAG, the loss function can be generalised in two different ways. One can penalise a mistake if *all* ancestor nodes are predicted correctly (in this case, the above definition carries over), or one can penalise a mistake if there *exists* a correctly predicted path from the root to the node.

In the rest of the text, we do not consider this evaluation function, since it requires thresholded predictions, and we are interested in evaluating our methods regardless of any threshold.

### 3.5.2 Precision-recall based evaluation

As argued before, we wish to evaluate our predictive models independently from the threshold, as different contexts may require different threshold settings. Generally, in the binary case, two types of evaluation are suitable for this: ROC analysis [Provost and Fawcett, 1998] and analysis of precision-recall curves (PR curves). While ROC analysis is probably better known in the machine learning community, in our case PR analysis is more suitable. We will explain why this is so in a moment, first we define PR curves.

A precision-recall curve (PR curve) plots the precision of a model as a function of its recall. Note that these measures ignore the number of correctly predicted negative examples. Assume the model predicts the probability that a new instance is positive, and that we threshold this probability with a threshold $t$ to obtain the predicted class. A given threshold corresponds to a single point in PR space, and by varying the threshold we obtain a PR curve: while decreasing $t$ from 1.0 to 0.0, an increasing number of instances is predicted positive, causing the recall to increase whereas precision may increase or decrease (with normally a tendency to decrease).

Although a PR curve helps in understanding the predictive behavior of the model, a single performance score is more useful to compare models. A score often used to this end is the area between the PR curve and the recall axis, the so-called "area under the PR curve" (AUPRC). The closer the AUPRC is to 1.0, the better the model is.

The reason why we believe PR curves to be a more suitable evaluation measure in this context is the following. In HMC datasets, it is often the case that individual classes have few positive instances. For example, in functional genomics, typically only a few genes have a particular function. This implies that for most classes, the

number of negative instances by far exceeds the number of positive instances. We are more interested in recognising the positive instances (that an instance has a given label), rather than correctly predicting the negative ones (that an instance does not have a particular label). Although ROC curves are better known, we believe that they are less suited for this task, exactly because they reward a learner if it correctly predicts negative instances (giving rise to a low false positive rate). This can present an overly optimistic view of the algorithm's performance. This effect has been convincingly demonstrated and studied by Davis and Goadrich [2006], and we refer to them for further details.

A final point to note is that PR curves can be constructed for each individual class in a multi-label classification task by taking as positives the examples belonging to the class and as negatives the other examples. How to combine the class-wise performances in order to quantify the overall performance, is less straightforward. The following two paragraphs discuss two approaches, each of which are meaningful.

#### 3.5.2.1 Area under the average PR curve

A first approach to obtain an overall performance score is to construct an overall PR curve by transforming the multi-label problem into a binary problem as follows [Yang, 1999; Tsoumakas and Vlahavas, 2007; Blockeel et al., 2006]. Consider a binary classifier that takes as input an instance-class couple and predicts whether that instance belongs to that class or not. Precision is then the proportion of positively predicted couples that are positive and recall is the proportion of positive couples that are correctly predicted positive. A rank classifier (which predicts how likely it is that the instance belongs to the class) can be turned into such a binary classifier by choosing a threshold, and by varying this threshold a PR curve is obtained. We will evaluate our predictive model in exactly the same way as such a rank classifier.

For a given threshold value, this yields one point $(\overline{Precision}, \overline{Recall})$ in PR space, which can be computed as follows:

$$\overline{Precision} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FP_i}, \quad \text{and} \quad \overline{Recall} = \frac{\sum_i TP_i}{\sum_i TP_i + \sum_i FN_i},$$

where $i$ ranges over all classes. This corresponds to micro-averaging the precision and recall. In terms of the original problem definition, $\overline{Precision}$ corresponds to the proportion of predicted labels that are correct and $\overline{Recall}$ to the proportion of labels in the data that are correctly predicted.

By varying the threshold, we obtain an average PR curve. We denote the area under this curve with AU($\overline{\text{PRC}}$).

#### 3.5.2.2 Average area under the PR curves

A second approach is to take the (weighted) average of the areas under the individual (per class) PR curves, computed as follows:

$$\overline{AUPRC}_{w_1,\ldots,w_{|C|}} = \sum_i w_i \cdot AUPRC_i$$

The most obvious instantiation of this approach is to set all weights to $1/|C|$, with $C$ the set of classes. In the results, we denote this measure with $\overline{\text{AUPRC}}$. A second instantiation is to weigh the contribution of a class with its frequency, that is, $w_i = v_i / \sum_j v_j$, with $v_i$ $c_i$'s frequency in the data. The rationale behind this is that for some applications more frequent classes may be more important. We denote the latter measure with $\overline{\text{AUPRC}}_{\text{w}}$.

A corresponding PR curve that has precisely $\overline{AUPRC}_{w_1,\ldots,w_{|C|}}$ as area can be drawn by taking, for each value on the recall axis, the (weighted) point-wise average of the class-wise precision values. Note that the interpretation of this curve is different from that of the micro-averaged PR curve defined in the previous section. For example, each point on this curve may correspond to a different threshold for each class. Section 3.6.3.4 presents examples of both types of curves and provides more insight in the difference in interpretation.

## 3.6 Experiments in yeast functional genomics

In this section we give an experimental answer to the following questions:

- How does Clus-HMC, Clus-SC and Clus-HSC relate to one another with respect to predictive performance, model size and computational efficiency (time needed for learning and applying the model), both on a tree-shaped and a DAG-shaped hierarchy?

- In earlier work [Struyf et al., 2005] it was assumed that it is sensible to use greater weights in Clus-HMC's distance measure for classes higher up in the hierarchy. This was not validated experimentally, however. What is the effect of using different weighting schemes?

- For multi-label classification, it is not obvious how to measure the overall performance of a predictive system, averaged out over all classes. What is the relation between the proposed evaluation measures and do they have specific advantages and disadvantages?

Before presenting the results (Sect. 3.6.3), we first discuss the datasets used in our evaluation (Sect. 3.6.1) and the applied methodology (Sect. 3.6.2).

Table 3.2: Dataset properties: number of instances $|D|$, number of attributes $|A|$.

| | Dataset | $\|D\|$ | $\|A\|$ |
|---|---|---|---|
| $D_1$ | Sequence Clare [2003] (seq) | 3932 | 478 |
| $D_2$ | Phenotype Clare [2003] (pheno) | 1592 | 69 |
| $D_3$ | Secondary structure Clare [2003] (struc) | 3851 | 19628 |
| $D_4$ | Homology search Clare [2003] (hom) | 3867 | 47034 |
| $D_5$ | Spellman et al. [1998] (cellcycle) | 3766 | 77 |
| $D_6$ | Roth et al. [1998] (church) | 3764 | 27 |
| $D_7$ | DeRisi et al. [1997] (derisi) | 3733 | 63 |
| $D_8$ | Eisen et al. [1998] (eisen) | 2425 | 79 |
| $D_9$ | Gasch et al. [2000] (gasch1) | 3773 | 173 |
| $D_{10}$ | Gasch et al. [2001] (gasch2) | 3788 | 52 |
| $D_{11}$ | Chu et al. [1998] (spo) | 3711 | 80 |
| $D_{12}$ | All microarray Clare [2003] (expr) | 3788 | 551 |

### 3.6.1 Datasets

*Saccharomyces cerevisiae* (baker's or brewer's yeast) is one of biology's classic model organisms, and has been the subject of intensive study for years.

We use 12 yeast datasets from Clare [2003] (Table 3.2), but with new and updated class labels. The different datasets describe different aspects of the genes in the yeast genome. They include five types of bioinformatic data: sequence statistics, phenotype, secondary structure, homology, and expression. The different sources of data highlight different aspects of gene function. Below, we describe each dataset in turn.

**$D_1$** (seq) records sequence statistics that depend on the amino acid sequence of the protein for which the gene codes. These include amino acid ratios, sequence length, molecular weight and hydrophobicity. Some of the properties were calculated using ProtParam Expasy [2008], some were taken from MIPS [Mewes et al., 1999] (e.g., the gene's chromosome number), and some were simply calculated directly. $D_1$'s attributes are mostly real valued, although some (like chromosome number or strand) are discrete.

**$D_2$** (pheno) contains phenotype data, which represents the growth or lack of growth of knock-out mutants that are missing the gene in question. The gene is removed or disabled and the resulting organism is grown with a variety of media to determine what the modified organism might be sensitive or resistant to. This data was taken from EUROFAN, MIPS and TRIPLES [Oliver, 1996; Mewes et al., 1999; Kumar et al., 2000]. The attributes are discrete, and the dataset is sparse, since not all knock-outs have been grown under all conditions.

**$D_3$** (struc) stores features computed from the secondary structure of the yeast proteins. The secondary structure is not known for all yeast genes; however, it can

Table 3.3: Properties of the two classification schemes. $|C|$ is the average number of classes actually used in the datasets (out of the total number of classes defined by the scheme). $|S|$ is the average number of labels per example, with between parentheses the average number counting only the most specific classes of an example.

| | FunCat | GO |
|---|---|---|
| Scheme version | 2.1 (2007/01/09) | 1.2 (2007/04/11) |
| Yeast annotations | 2007/03/16 | 2007/04/07 |
| Total classes | 1362 | 22960 |
| Dataset average $\|C\|$ | 492 (6 levels) | 3997 (14 levels) |
| Dataset average $\|S\|$ | 8.8 (3.2 most specific) | 35.0 (5.0 most specific) |

be predicted from the protein sequence with reasonable accuracy. The program PROF [Ouali and King, 2000] was used to this end. Due to the relational nature of secondary structure data, Clare performed a preprocessing step of relational frequent pattern mining; $D_3$ includes the constructed patterns as binary attributes.

$\mathbf{D_4}$ (hom) includes for each yeast gene, information from other, homologous genes. Homology is usually determined by sequence similarity. PSI-BLAST [Altschul et al., 1997] was used to compare yeast genes both with other yeast genes, and with all genes indexed in SwissProt 39. This provided for each yeast gene, a list of homologous genes. For each of these, various properties were extracted (keywords, sequence length, names of databases they are listed in, ...). Clare preprocessed this data in a similar way as the secondary structure data, to produce binary attributes.

$\mathbf{D_5}, \ldots, \mathbf{D_{12}}$. The use of microarrays to record the expression of genes is popular in biology and bioinformatics. Microarray chips provide the means to test the expression levels of genes across an entire genome in a single experiment. Many expression datasets exist for yeast, and several of these were used. Attributes for these datasets are real valued, representing fold changes in expression levels.

We construct two versions of each dataset. The input attributes are identical in both versions, but the classes are taken from two different classification schemes. In the first version, they are from FunCat (`http://mips.gsf.de/projects/funcat`), a scheme for classifying the functions of gene products, developed by MIPS [Mewes et al., 1999]. FunCat is a tree-structured class hierarchy; a small part is shown in Fig. 3.1. In the second version of the datasets, the genes are annotated with terms from the Gene Ontology (GO) [Ashburner et al., 2000] (`http://www.geneontology.org`), which forms a directed acyclic graph instead of a tree: each term can have multiple parents (we use GO's "is-a" relationship between terms)[3]. Table 3.3 compares the properties of FunCat

[3]The GO versions of the datasets may contain slightly fewer examples, since not all genes in

and GO. Note that GO has an order of magnitude more classes than FunCat for our datasets. The 24 resulting datasets can be found on the following webpage `http://www.cs.kuleuven.be/~dtai/clus/hmcdatasets.html`.

### 3.6.2 Method

Clare [2003] presents models trained on 2/3 of each dataset and tested on the remaining 1/3. In our experiments we use exactly the same training and test sets.

The stopping criterion (i.e., the function Acceptable in Table 3.1) was implemented as follows. The minimal number of examples a leaf has to cover was set to 5 for all algorithms. The F-test that is used to check the significance of the variance reduction takes a significance level parameter $s$, which was optimised as follows: for each out of 6 available values for $s$, CLUS-HMC was run on 2/3 of the training set and its PR curve for the remaining 1/3 validation set was constructed. The $s$ parameter yielding the largest area under this average validation PR curve was then used to train the model on the complete training set. This optimisation was performed independently for each evaluation measure (discussed in Sect. 3.5) and each weighting scheme (Sect. 3.4.1). The PR curves or AUPRC values that are reported are obtained by testing the resulting model on the test set. The results for CLUS-SC and CLUS-HSC were obtained in the same way as for CLUS-HMC, but with a separate run for each class (including separate optimisation of $s$ for each class).

We compare the AUPRC of the different methods, using the approaches discussed in Sect. 3.5. For GO, which consists of three separate hierarchies, we left out the three classes representing these hierarchies' roots, since they occur for all examples. PR curves were constructed with proper (non-linear) interpolation between points, as described by Davis and Goadrich [2006]. The non-linearity comes from the fact that precision is non-linear in the number of true positives and false positives.

To estimate significance of the AUPRC comparison, we use the (two-sided) Wilcoxon signed rank test [Wilcoxon, 1945], which is a non-parametric alternative to the paired Student's t-test that does not make any assumption about the distribution of the measurements. In the results, we report the $p$-value of the test and the corresponding rank sums.[4]

The experiments were run on a cluster of AMD Opteron processors (1.8 - 2.4GHz, >2GB RAM) running Linux.

---

the original datasets are annotated with GO terms.

[4]The Wilcoxon test compares two methods by ranking the pairwise differences in their performances by absolute value. Then it calculates the sums for the ranks corresponding to positive and negative differences. The smaller of these two rank sums is compared to a table of all possible distributions of ranks to calculate $p$.

### 3.6.3 Results

In the experiments, we are dealing with many dimensions: we have 12 different descriptions of gene aspects and 2 class hierarchies, resulting in 24 datasets with several hundreds of classes each, on which we want to compare 3 algorithms. Moreover, we consider 3 precision-recall evaluation measures, and for Clus-HMC we proposed 4 weighting schemes. In order to deal with this complex structure, we proceed as follows.

We start by evaluating the different weighting schemes used in the Clus-HMC algorithm. Then we compare the predictive performance of the three algorithms Clus-HMC, Clus-HSC, and Clus-SC. Next, we study the relation between the three evaluation measures AU($\overline{\text{PRC}}$), $\overline{\text{AUPRC}}$, and $\overline{\text{AUPRC}}_{\text{w}}$. Afterwards, we give some example PR curves for specific datasets. Finally, we compare the model size and induction times of the three algorithms.

#### 3.6.3.1 Comparison of weighting schemes

First, we investigate different instantiations for the weights in the weighted Euclidean distance metric used in the heuristic of Clus-HMC. We have arbitrarily set $w_0$ to 0.75. The precise questions that we want to answer are:

1. Is it useful to use weights in Clus-HMC's heuristic? In other words, is there a difference between using weights that decrease with the hierarchy depth and setting all weights to 1.0?

2. If yes, which of the weighting schemes for combining the weights of multiple parents (Sect. 3.4.1) yields the best results for datasets with DAG structured class labels?

Tables 3.4 (upper part) and 3.5 show the average AUPRC values, and the Wilcoxon test outcomes for FunCat. As can be seen from the tables, using weights has slight advantages over not using weights. Therefore, for FunCat only results using weights will be reported in the rest of the paper. Recall that FunCat is a tree hierarchy, and thus, the second question does not apply.

For GO, the results are less clear. Table 3.4 (lower part) shows the average AUPRC values and Fig. 3.5 visualises the Wilcoxon outcomes. W.r.t. $\overline{\text{AUPRC}}_{\text{w}}$, there are no differences between the methods. For AU($\overline{\text{PRC}}$), $w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$ performs slightly better than all other methods (although not significant), while for $\overline{\text{AUPRC}}$, $w(c) = w_0 \cdot \max_j w(par_j(c))$ performs better. For the rest of the experiments, we decided to use the former because it also performs well for $\overline{\text{AUPRC}}$ and because the averaging may make the scheme more robust than the scheme that takes the parents' weights maximum.

Table 3.4: Weighting schemes for FunCat and GO: AU($\overline{\text{PRC}}$), $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ averaged over all datasets. (90% confidence intervals are indicated after the '$\pm$' sign.)

| FunCat | AU($\overline{\text{PRC}}$) | $\overline{\text{AUPRC}}$ | $\overline{\text{AUPRC}}_{\text{w}}$ |
|---|---|---|---|
| 1.0 | 0.191 $\pm$ 0.012 | 0.042 $\pm$ 0.006 | 0.162 $\pm$ 0.015 |
| $w_0 \cdot w(par_j(c))$ | 0.194 $\pm$ 0.013 | 0.045 $\pm$ 0.009 | 0.164 $\pm$ 0.017 |
| GO | AU($\overline{\text{PRC}}$) | $\overline{\text{AUPRC}}$ | $\overline{\text{AUPRC}}_{\text{w}}$ |
| 1.0 | 0.364 $\pm$ 0.011 | 0.028 $\pm$ 0.005 | 0.342 $\pm$ 0.015 |
| $w_0 \sum_j w(par_j(c))$ | 0.364 $\pm$ 0.010 | 0.028 $\pm$ 0.005 | 0.342 $\pm$ 0.015 |
| $w_0 \cdot \min_j w(par_j(c))$ | 0.365 $\pm$ 0.009 | 0.027 $\pm$ 0.005 | 0.342 $\pm$ 0.014 |
| $w_0 \cdot \text{avg}_j w(par_j(c))$ | 0.365 $\pm$ 0.009 | 0.028 $\pm$ 0.005 | 0.342 $\pm$ 0.013 |
| $w_0 \cdot \max_j w(par_j(c))$ | 0.364 $\pm$ 0.009 | 0.028 $\pm$ 0.005 | 0.342 $\pm$ 0.013 |

Table 3.5: Weighting schemes for FunCat: comparing $w(c) = w_0 \cdot w(par(c))$ to $w(c) = 1.0$. A '$\oplus$' means that $w(c) = w_0 \cdot w(par(c))$ performs better than $w(c) = 1.0$ according to the Wilcoxon signed rank test. The table indicates the rank sums and corresponding $p$-values computed by the test.

| | $w(c) = w_0 \cdot w(par(c))$ | |
|---|---|---|
| FunCat | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\oplus$ 51/15 | 0.12 |
| $\overline{\text{AUPRC}}$ | $\oplus$ 61/17 | 0.09 |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\oplus$ 53/25 | 0.30 |

**Conclusion** Earlier it was assumed that it is advisable to use weights in the calculation of Clus-HMC's distance measure, giving greater weights to classes appearing higher in the hierarchy. However, it turns out that using weights is only slightly better than not using weights, while the improvement is not statistically significant. For GO, averaging the weights of the parent nodes seems the best option. Recall that for tree shaped hierarchies, the DAG weighting schemes all become identical to the tree weighting scheme. As a result, we can use the same weighting method ($w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$) for both the GO and FunCat experiments.

#### 3.6.3.2 Precision-recall based comparison of Clus-HMC/SC/HSC

In Sect. 3.3.6, we already mentioned that we expect Clus-HMC to run more efficiently than Clus-SC and Clus-HSC, to deal better with imbalanced class distributions and to identify global features while obeying the hierarchy constraint. One would further hope that these advantages do not come at the cost of worse

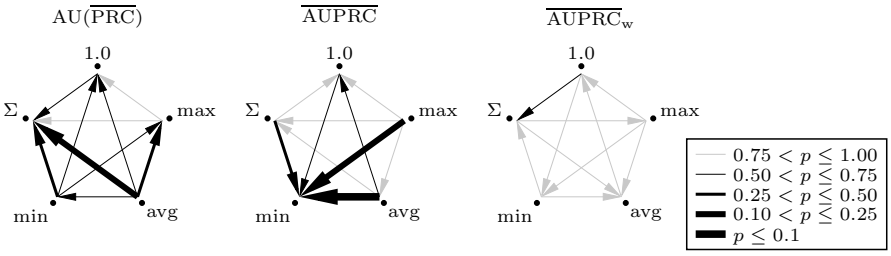


Figure 3.5: Weighting schemes for GO: $w(c) = 1.0$, $w(c) = w_0 \sum_j w(par_j(c))$, $w(c) = w_0 \cdot \min_j w(par_j(c))$, $w(c) = w_0 \cdot \text{avg}_j w(par_j(c))$, $w(c) = w_0 \cdot \max_j w(par_j(c))$. An arrow from scheme $A$ to $B$ indicates that $A$ is better than $B$. The line width of the arrow indicates the significance of the difference according to the Wilcoxon signed rank test.

predictive performance.

Tables 3.6 and 3.7 show AUPRC values for the three algorithms for FunCat and GO, respectively. Summarising Wilcoxon outcomes comparing Clus-HMC to Clus-SC and Clus-HSC are shown in Table 3.8. We see that Clus-HMC performs better than Clus-SC and Clus-HSC, both for FunCat and GO, and for all evaluation measures. This is unexpected: one would think that Clus-SC or Clus-HSC has the advantage because it can learn a different optimal model for each class. Our original conjecture was that this was due to Clus-HMC performing better on the lower levels of the hierarchy.[5] But a per-level computation of the predictive performance does not confirm this: Clus-HMC tends to perform better overall. There is no clear correlation with depth in the hierarchy.

Table 3.9 compares Clus-HSC to Clus-SC. Clus-HSC performs better than Clus-SC on GO, w.r.t. all evaluation measures. On FunCat, Clus-HSC is better than Clus-SC w.r.t. AU($\overline{\text{PRC}}$). According to the two other evaluation measures, Clus-SC performs better, but the difference is not significant.

**Conclusion** Clus-HMC performs better than Clus-SC for both FunCat and GO hierarchies. Clus-HMC also outperforms Clus-HSC in both settings. Clus-HSC in turn outperforms Clus-SC on GO. For FunCat, the results depend on the evaluation measure, and the differences are not significant.

[5]Level four classes are very infrequent and therefore difficult to learn, but in Clus-HMC the parent classes may help in keeping the instances from class x/y/z together in the tree and within a node with mainly x/y/z instances, a class x/y/z/u has higher relative frequency.

Table 3.6: Predictive performance (AUPRC) of the different algorithms for FunCat.

| | AU($\overline{\text{PRC}}$) | | | $\overline{\text{AUPRC}}$ | | | $\overline{\text{AUPRC}}_{\text{w}}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | HMC | HSC | SC | HMC | HSC | SC | HMC | HSC | SC |
| seq | 0.211 | 0.091 | 0.095 | 0.053 | 0.043 | 0.042 | 0.183 | 0.151 | 0.154 |
| pheno | 0.160 | 0.152 | 0.149 | 0.030 | 0.031 | 0.031 | 0.124 | 0.125 | 0.127 |
| struc | 0.181 | 0.118 | 0.114 | 0.041 | 0.039 | 0.040 | 0.161 | 0.152 | 0.152 |
| hom | 0.254 | 0.155 | 0.153 | 0.089 | 0.067 | 0.076 | 0.240 | 0.205 | 0.205 |
| cellcycle | 0.172 | 0.111 | 0.106 | 0.034 | 0.036 | 0.038 | 0.142 | 0.146 | 0.146 |
| church | 0.170 | 0.131 | 0.128 | 0.029 | 0.029 | 0.031 | 0.129 | 0.127 | 0.128 |
| derisi | 0.175 | 0.094 | 0.089 | 0.033 | 0.029 | 0.028 | 0.137 | 0.125 | 0.122 |
| eisen | 0.204 | 0.127 | 0.132 | 0.052 | 0.052 | 0.055 | 0.183 | 0.169 | 0.173 |
| gasch1 | 0.205 | 0.106 | 0.104 | 0.049 | 0.047 | 0.047 | 0.176 | 0.154 | 0.153 |
| gasch2 | 0.195 | 0.121 | 0.119 | 0.039 | 0.042 | 0.037 | 0.156 | 0.148 | 0.147 |
| spo | 0.186 | 0.103 | 0.098 | 0.035 | 0.038 | 0.034 | 0.153 | 0.139 | 0.139 |
| expr | 0.210 | 0.127 | 0.123 | 0.052 | 0.054 | 0.050 | 0.179 | 0.167 | 0.167 |
| Average: | 0.194 | 0.120 | 0.118 | 0.045 | 0.042 | 0.042 | 0.164 | 0.151 | 0.151 |

Table 3.7: Predictive performance (AUPRC) of the different algorithms for GO.

| | AU($\overline{\text{PRC}}$) | | | $\overline{\text{AUPRC}}$ | | | $\overline{\text{AUPRC}}_{\text{w}}$ | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | HMC | HSC | SC | HMC | HSC | SC | HMC | HSC | SC |
| seq | 0.386 | 0.282 | 0.197 | 0.036 | 0.035 | 0.035 | 0.373 | 0.283 | 0.279 |
| pheno | 0.337 | 0.416 | 0.316 | 0.021 | 0.019 | 0.021 | 0.299 | 0.239 | 0.238 |
| struc | 0.358 | 0.353 | 0.228 | 0.025 | 0.026 | 0.026 | 0.328 | 0.266 | 0.262 |
| hom | 0.401 | 0.353 | 0.252 | 0.051 | 0.053 | 0.052 | 0.389 | 0.317 | 0.313 |
| cellcycle | 0.357 | 0.371 | 0.252 | 0.021 | 0.024 | 0.020 | 0.335 | 0.275 | 0.267 |
| church | 0.348 | 0.397 | 0.289 | 0.018 | 0.016 | 0.017 | 0.316 | 0.248 | 0.247 |
| derisi | 0.355 | 0.349 | 0.218 | 0.019 | 0.017 | 0.017 | 0.321 | 0.248 | 0.246 |
| eisen | 0.380 | 0.365 | 0.270 | 0.036 | 0.035 | 0.031 | 0.362 | 0.303 | 0.294 |
| gasch1 | 0.371 | 0.351 | 0.239 | 0.030 | 0.028 | 0.026 | 0.353 | 0.290 | 0.282 |
| gasch2 | 0.365 | 0.378 | 0.267 | 0.024 | 0.026 | 0.023 | 0.347 | 0.282 | 0.278 |
| spo | 0.352 | 0.371 | 0.213 | 0.026 | 0.020 | 0.020 | 0.324 | 0.254 | 0.254 |
| expr | 0.368 | 0.351 | 0.249 | 0.029 | 0.028 | 0.028 | 0.353 | 0.286 | 0.284 |
| Average: | 0.365 | 0.361 | 0.249 | 0.028 | 0.027 | 0.026 | 0.342 | 0.274 | 0.270 |

Table 3.8: Clus-HMC compared to Clus-SC and Clus-HSC. A '$\oplus$' ('$\ominus$') means that Clus-HMC performs better (worse) than the given method according to the Wilcoxon signed rank test. The table indicates the rank sums and corresponding $p$-values. Differences significant at the 0.01 level are indicated in bold.

| | HMC vs. SC | | HMC vs. HSC | |
|---|---|---|---|---|
| FunCat | Score | $p$ | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| $\overline{\text{AUPRC}}$ | $\oplus$ 51/27 | $3.8 \cdot 10^{-1}$ | $\oplus$ 43/35 | $7.9 \cdot 10^{-1}$ |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\oplus$ **73/5** | $\mathbf{4.9 \cdot 10^{-3}}$ | $\oplus$ **74/4** | $\mathbf{3.4 \cdot 10^{-3}}$ |
| GO | Score | $p$ | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ 43/35 | $7.9 \cdot 10^{-1}$ |
| $\overline{\text{AUPRC}}$ | $\oplus$ 68/10 | $2.1 \cdot 10^{-2}$ | $\oplus$ 55/23 | $2.3 \cdot 10^{-1}$ |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |

Table 3.9: Clus-HSC compared to Clus-SC. A '$\oplus$' ('$\ominus$') means that Clus-HSC performs better (worse) than Clus-SC according to the Wilcoxon signed rank test.

| | HSC vs. SC | |
|---|---|---|
| FunCat | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\oplus$ 62/16 | $7.7 \cdot 10^{-2}$ |
| $\overline{\text{AUPRC}}$ | $\ominus$ 37/41 | $9.1 \cdot 10^{-1}$ |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\ominus$ 22/56 | $2.0 \cdot 10^{-1}$ |

| | HSC vs. SC | |
|---|---|---|
| GO | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| $\overline{\text{AUPRC}}$ | $\oplus$ 49/29 | $4.7 \cdot 10^{-1}$ |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |

#### 3.6.3.3 Relation between the different AUPRC measures

In Sect. 3.5, we have proposed several ways of combining class-wise PR-curves into a single PR-curve. It turns out that these methods are quite different with respect to what they measure.

This difference is best explained by looking at the behavior of these curves for a default model, that is, a degenerate decision tree that consists of precisely one leaf (one Clus-HMC leaf, or equivalently, a set of single leaf Clus-SC trees). The class-wise predicted probabilities of 'default' are constant (the same for each test instance) and equal to the proportion of training instances in the corresponding class, i.e., the class frequency (Fig. 3.6a).

**PR curves of a default classifier** The PR-curve of this default predictor for a single class $c_i$ is as follows: if the overall frequency $f_i$ of the class is above $t$, then the predictor predicts positive for all instances, so we get a recall of 1 and a precision of $f_i$; otherwise it predicts negative for all instances, giving a recall

(a)

| default | |
|---|---|
| $c_i$ | $v_i$ |
| $c_1$ | 0.9 |
| $c_2$ | 0.5 |
| $c_3$ | 0.1 |

| allclass | |
|---|---|
| $c_i$ | $v_i$ |
| $c_1$ | 1.0 |
| $c_2$ | 1.0 |
| $c_3$ | 1.0 |

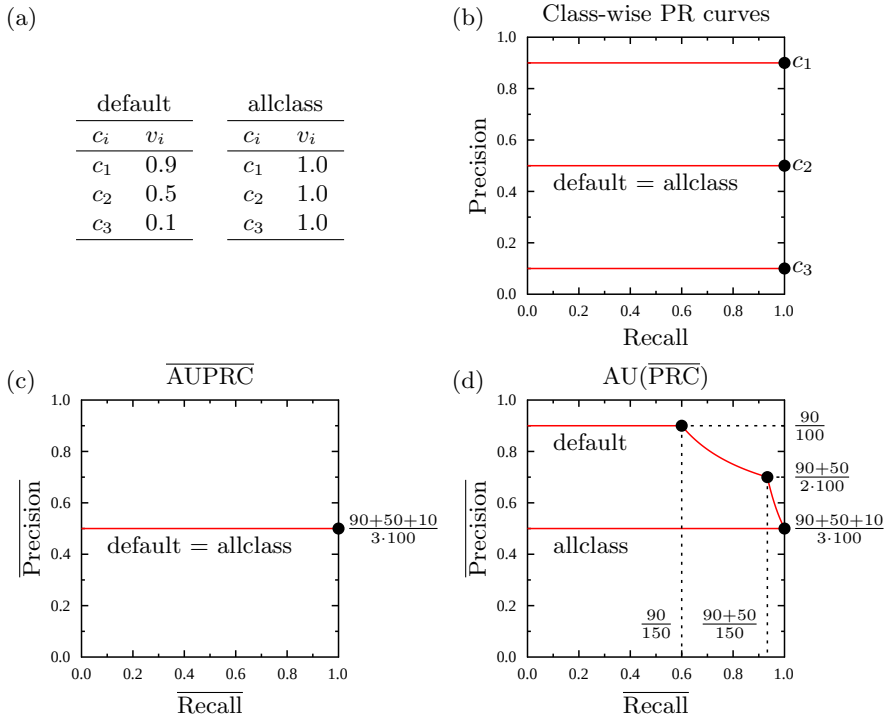


Figure 3.6: Example for a dataset with 100 instances. (a) Two degenerate decision tree models: 'default' and 'allclasses'. The 'default' model's predicted set of classes depends on the classification threshold $t$, while the 'allclasses' model predicts all classes independent of $t$. (b) Class-wise PR curves (identical for 'default' and 'allclasses'). (c) Average PR curve corresponding to $\overline{\text{AUPRC}}$. (d) Average PR curves corresponding to AU($\overline{\text{PRC}}$) (the non-linear curves connecting the points are obtained by means of proper PR interpolation [Davis and Goadrich, 2006]).

of 0 and an undefined precision. This leads to one point in the PR-diagram at $(1, f_i)$. To obtain a PR-curve, observe that randomly discarding a fraction of the predictions results in the same precision, but a smaller recall. The PR-curve thus becomes the horizontal line $(r, f_i)$ with $0 < r \leq 1$ (Fig. 3.6b) [Davis and Goadrich, 2006].

Consequently, the average PR-curve constructed using the $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ methods is also a horizontal line, at height $\frac{1}{|C|}\sum_i f_i$ or $\sum_i w_i f_i$, respectively. The former is shown in Fig. 3.6c.

The average PR-curve for AU($\overline{\text{PRC}}$) is quite different, though. This curve is constructed from predictions for all classes together. For a threshold $t$, all instances

Table 3.10: CLUS-SC, CLUS-HSC and CLUS-HMC compared to the default model. A '$\oplus$' ('$\ominus$') means that the given method performs better (worse) than default according to the Wilcoxon signed rank test.

| | SC vs. DEF | | HSC vs. DEF | | HMC vs. DEF | |
|---|---|---|---|---|---|---|
| FunCat | Score | $p$ | Score | $p$ | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\ominus$ **1/77** | $\mathbf{9.8 \cdot 10^{-4}}$ | $\ominus$ **2/76** | $\mathbf{1.5 \cdot 10^{-3}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| $\overline{\text{AUPRC}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| GO | Score | $p$ | Score | $p$ | Score | $p$ |
| AU($\overline{\text{PRC}}$) | $\ominus$ **0/78** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ 68/10 | $2.1 \cdot 10^{-2}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| $\overline{\text{AUPRC}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |
| $\overline{\text{AUPRC}}_{\text{w}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ | $\oplus$ **78/0** | $\mathbf{4.9 \cdot 10^{-4}}$ |

are assigned exactly the classes $S$ with frequency above $t$, i.e., $S = \{c_i \mid f_i \geq t\}$. While decreasing the classification threshold $t$ from 1.0 to 0.0, $S$ grows from the empty set to the set of all classes $C$. At the same time, the average precision drops from the frequency of the most frequent class to the average of the class frequencies. Correct interpolation between the points [Davis and Goadrich, 2006] leads to curves such as the one shown in Fig. 3.6d.

**Interpretation of different average default curves** Now consider the model 'allclasses' (Fig. 3.6a), that predicts each class with probability 1.0. This model's classwise PR curves are shown in Fig. 3.6b and are identical to those of 'default'. As a consequence, also the average PR curve combined with $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ is identical to those of 'default' (Fig. 3.6c). Since the set $S = C$ for all values of $t$ for 'allclasses', its average PR curve for AU($\overline{\text{PRC}}$) is a horizontal line with precision equal to the average of the class frequencies (Fig. 3.6d), just as for $\overline{\text{AUPRC}}$. These results show that it is more difficult to outperform 'default' with AU($\overline{\text{PRC}}$) than with $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$: in the latter cases, the model is better than default if it is better than always predicting all classes. Another way of stating this is that AU($\overline{\text{PRC}}$) rewards a predictor for exploiting information about the frequencies of the different classes. The $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ methods, on the other hand, average the performance of individual classes, i.e., they ignore the predictor's ability to learn the class frequencies.

**Comparison of Clus-HMC/SC/HSC to default** Table 3.10 compares CLUS-HMC, SC, and HSC to the default model. W.r.t. $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$, all models perform better than 'default', and this is true for all 24 datasets. This means that on average, for each individual class, the models perform better than always predicting the class. Interestingly, if we consider AU($\overline{\text{PRC}}$), then CLUS-SC,

Table 3.11: Difference between AUPRC obtained on the training set and AUPRC obtained on the test set. A higher (lower) difference indicates more (less) overfitting.

| | FunCat | | | GO | | |
|---|---|---|---|---|---|---|
| | HMC | HSC | SC | HMC | HSC | SC |
| AU($\overline{\text{PRC}}$) | 0.034 | 0.435 | 0.464 | 0.027 | 0.293 | 0.402 |
| $\overline{\text{AUPRC}}$ | 0.075 | 0.248 | 0.267 | 0.045 | 0.218 | 0.190 |
| $\overline{\text{AUPRC}}_{\text{w}}$ | 0.109 | 0.375 | 0.389 | 0.061 | 0.317 | 0.308 |

and also Clus-HSC on FunCat, perform worse than 'default'. W.r.t. this evaluation measure, these methods produce overly complex models and may overfit the training data.

The overfitting can be quantified by subtracting the AUPRC obtained on the test set from that on the training set. Table 3.11 shows these differences, which are indeed highest for the Clus-SC and Clus-HSC methods.

**Conclusion** We have shown that the three proposed ways of averaging classwise PR curves are indeed different. The AU($\overline{\text{PRC}}$) evaluation measure looks at the performance of the model in a mix of classes, whereas the $\overline{\text{AUPRC}}$ and $\overline{\text{AUPRC}}_{\text{w}}$ measures evaluate the performance of individual classes independently. W.r.t. the AU($\overline{\text{PRC}}$) measure, Clus-SC, and also Clus-HSC on FunCat, were shown to overfit the training data.

#### 3.6.3.4 Example PR curves for specific datasets

Figures 3.7 and 3.8 show averaged PR curves for two datasets. These curves illustrate the above conclusions. We see that, for both datasets, Clus-HMC performs best, especially for the AU($\overline{\text{PRC}}$) or $\overline{\text{AUPRC}}_{\text{w}}$ evaluation measures. If we consider AU($\overline{\text{PRC}}$), then overfitting can be detected for Clus-SC and Clus-HSC.

Figure 3.9 shows a number of class-wise PR curves for the dataset 'hom'. We have chosen the four classes for which Clus-HMC (parameter $s$ optimised for AU($\overline{\text{PRC}}$)) yields the largest AUPRC on the validation set, compared to the default AUPRC for that class. Since not all of these classes occurred in the test set, we have chosen the four best classes that occur in at least 5% of the test examples. We see that for FunCat, Clus-HMC performs better on these classes, while for GO, the results are less clear. Indeed, if we look at Table 3.7, we see that the three algorithms perform similarly for this dataset if all classes are considered equally important (corresponding to the $\overline{\text{AUPRC}}$ evaluation method). However, the other evaluation methods (which do take into account class frequencies) show a higher gain for Clus-HMC, which indicates that the latter performs better on the

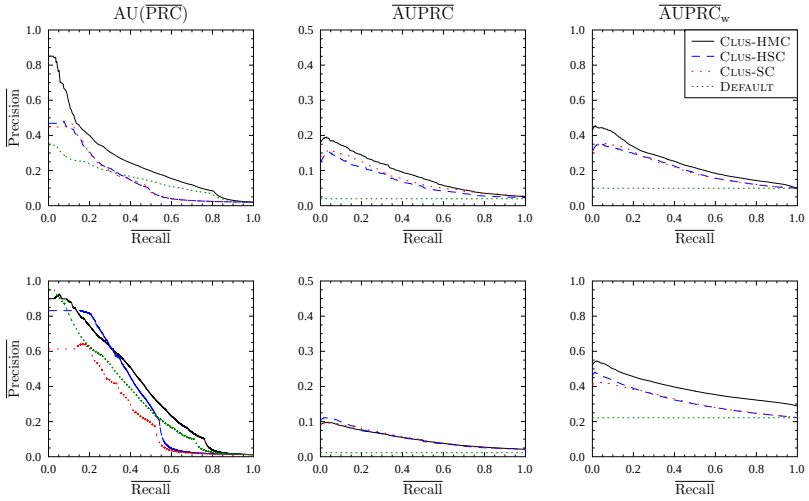


Figure 3.7: PR curves averaged over all classes according to the 3 evaluation measures for FunCat (top) and GO (bottom) for the dataset 'hom'.

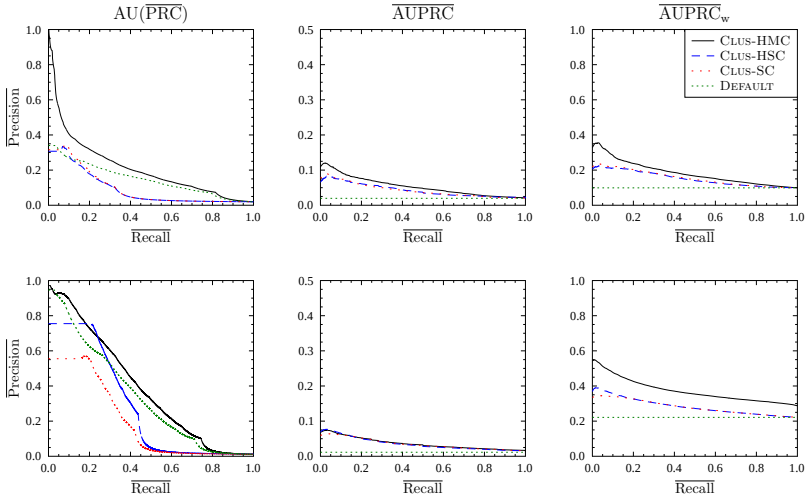


Figure 3.8: PR curves averaged over all classes according to the 3 evaluation measures for FunCat (top) and GO (bottom) for the dataset 'seq'.

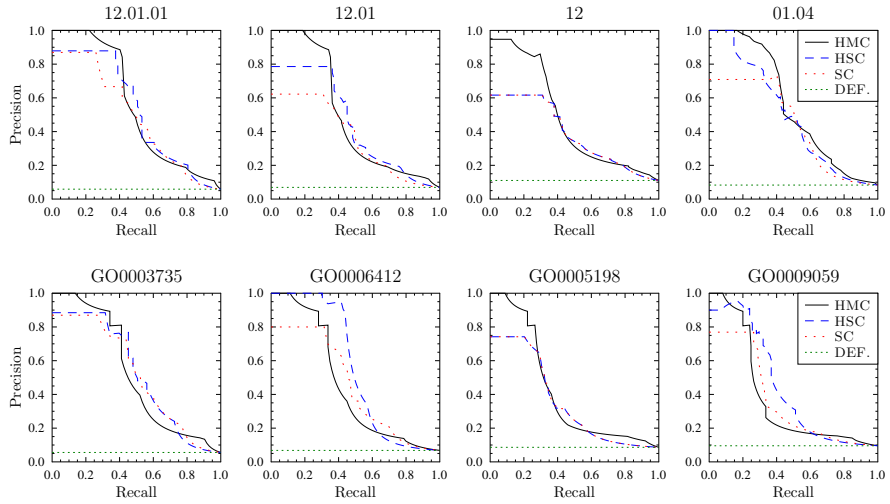


Figure 3.9: Example class-wise PR curves for FunCat (top) and GO (bottom) for the dataset 'hom'.

more frequent classes. Plotting the difference in AUPRC against class frequency (Fig. 3.10) confirms this result.

#### 3.6.3.5 Comparison of Clus-HMC/SC/HSC's tree size and induction time

We conclude this experimental evaluation by comparing the model size and computational efficiency of the three algorithms.

Tables 3.12 and 3.13 present the tree sizes obtained with the different methods. We measure tree size as the number of leaves. The tables include three numbers for Clus-HMC: one number for each of the evaluation measures. Recall that Clus-HMC uses the evaluation measure to tune its $F$-test parameter $s$. Different evaluation measures may yield different optimal $s$ values and therefore different trees. SC and HSC trees, on the other hand, predict only one class, so there is no need to average PR curves; the tree induction algorithm tunes its $F$-test parameter to maximise its class's AUPRC.

Averaged over the evaluation measures and datasets, the HMC trees contain 60.9 (FunCat) and 53.6 (GO) leaves. The SC trees, on the other hand, are smaller because they each model only one class. They include on average 15.9 (FunCat) and 7.6 (GO) leaves. Nevertheless, the total size of all SC trees is on average a factor 311.2 (FunCat) and 1049.8 (GO) larger than the corresponding HMC tree. This difference is bigger for GO than for FunCat because GO has an order of

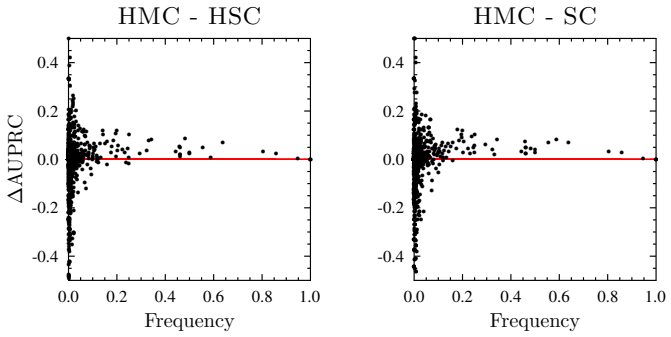


Figure 3.10: Difference in AUPRC versus class frequency for GO for the dataset 'hom'.

magnitude more classes (Table 3.3) and therefore also an order of magnitude more SC trees. Comparing HMC to HSC yields similar conclusions.

Observe that the HSC trees are smaller than the SC trees (a factor 2.2 on FunCat and 2.8 on GO). We see two reasons for this. First, HSC trees encode less knowledge than SC ones because they are conditioned on their parent class. That is, if a given feature subset is relevant to all classes in a sub-lattice of hierarchy, then Clus-SC must include this subset in each tree of the sub-lattice, while Clus-HSC

Table 3.12: Tree size (number of tree leaves) for FunCat.

| | Clus-HMC | | | Clus-SC | | Clus-HSC | |
|---|---|---|---|---|---|---|---|
| Dataset | $\overline{AU(PRC)}$ | $\overline{AUPRC}$ | $\overline{AUPRC}_w$ | Total | Average | Total | Average |
| seq | 14 | 168 | 168 | 10443 | 20.9 | 4923 | 9.9 |
| pheno | 8 | 8 | 8 | 1238 | 2.7 | 777 | 1.7 |
| struc | 12 | 125 | 56 | 8657 | 17.3 | 3917 | 7.8 |
| hom | 75 | 190 | 75 | 9137 | 18.3 | 4289 | 8.6 |
| cellcycle | 24 | 61 | 61 | 9671 | 19.4 | 4037 | 8.1 |
| church | 17 | 17 | 17 | 4186 | 8.4 | 2221 | 4.5 |
| derisi | 4 | 68 | 68 | 7807 | 15.6 | 3520 | 7.1 |
| eisen | 29 | 55 | 55 | 6311 | 13.7 | 2995 | 6.5 |
| gasch1 | 10 | 96 | 96 | 10447 | 20.9 | 4761 | 9.5 |
| gasch2 | 26 | 101 | 101 | 7850 | 15.7 | 3756 | 7.5 |
| spo | 6 | 43 | 43 | 8527 | 17.1 | 3623 | 7.3 |
| expr | 12 | 161 | 116 | 10262 | 20.6 | 4711 | 9.4 |
| Average: | 19.8 | 91.1 | 72.0 | 7878 | 15.9 | 3628 | 7.3 |

Table 3.13: Tree size (number of tree leaves) for GO.

| | CLUS-HMC | | | CLUS-SC | | CLUS-HSC | |
|---|---|---|---|---|---|---|---|
| Dataset | AU($\overline{\text{PRC}}$) | $\overline{\text{AUPRC}}$ | $\overline{\text{AUPRC}}_{\text{w}}$ | Total | Average | Total | Average |
| seq | 15 | 206 | 108 | 38969 | 9.4 | 21703 | 3.7 |
| pheno | 6 | 6 | 6 | 6213 | 2.0 | 5691 | 1.3 |
| struc | 14 | 76 | 76 | 36356 | 8.8 | 19147 | 3.3 |
| hom | 51 | 135 | 135 | 35270 | 8.5 | 19804 | 3.4 |
| cellcycle | 21 | 63 | 43 | 36260 | 8.8 | 19085 | 3.3 |
| church | 7 | 21 | 21 | 16049 | 3.9 | 12368 | 2.1 |
| derisi | 10 | 38 | 10 | 31175 | 7.6 | 16693 | 2.9 |
| eisen | 37 | 68 | 68 | 24844 | 7.0 | 14384 | 2.9 |
| gasch1 | 30 | 129 | 30 | 37838 | 9.2 | 20070 | 3.4 |
| gasch2 | 27 | 62 | 62 | 34204 | 8.3 | 18546 | 3.2 |
| spo | 14 | 60 | 60 | 35400 | 8.6 | 15552 | 2.7 |
| expr | 35 | 145 | 35 | 38313 | 9.3 | 20812 | 3.6 |
| Average: | 22.2 | 84.1 | 54.5 | 30908 | 7.6 | 16988 | 3.0 |

only needs them in the trees for the sub-lattice's most general border. Second, HSC trees use fewer training examples than SC trees, and tree size typically grows with training set size.

We also measure the total induction time for all methods. This is the time for building the actual trees; it does not include the time for loading the data and tuning the $F$-test parameter. CLUS-HMC requires on average 3.3 (FunCat) and 24.4 (GO) minutes to build a tree. CLUS-SC is a factor 58.6 (FunCat) and 129.0 (GO) slower than CLUS-HMC. CLUS-HSC is a factor 10.2 (FunCat) and 5.1 (GO) faster than CLUS-SC, but still a factor 6.3 (FunCat) and 55.9 (GO) slower than CLUS-HMC.

**Conclusion** Whereas the size of the individual trees learned by CLUS-HSC and CLUS-SC is smaller than the size of the trees output by CLUS-HMC, the total model size of the latter is much smaller than the total size of the models output by the single-label tree learners. As was expected, the CLUS-HSC models are smaller than the CLUS-SC models. Also w.r.t. efficiency, CLUS-HMC outperforms the other methods.

## 3.7 Conclusions

In this chapter, we have compared three decision tree algorithms on the task of hierarchical multi-label classification: (1) an algorithm that learns a single tree that predicts all classes at once (CLUS-HMC), (2) an algorithm that learns a separate

decision tree for each class (CLUS-SC), and (3) an algorithm that learns and applies such single-label decision trees in a hierarchical way (CLUS-HSC). The three algorithms are instantiations of the predictive clustering tree framework [Blockeel et al., 1998] and are designed for problems where the class hierarchy is either structured as a tree or as a directed acyclic graph (DAG). To our knowledge, the latter setting has not been studied before, although it occurs in real life applications. For instance, the Gene Ontology (GO), a widely used classification scheme for genes, is structured as a DAG. The DAG structure poses a number of complications to the algorithms e.g., the depth of a class in the hierarchy is no longer unique.

We have evaluated the algorithms on 24 datasets from functional genomics. The predictive performance was measured as area under the PR curve. For a single-label classification task this measure is well-defined, but for a multi-label problem the definition needs to be extended and there are several ways to do so. We propose three ways to construct a PR curve for the multi-label case: micro-averaging precision and recall for varying thresholds, taking the point-wise average of class-wise precision values for each recall value, and weighing the contribution of each class in this average by the class's frequency.

The most important results of our empirical evaluation are as follows. First, CLUS-HMC has a better predictive performance than CLUS-SC and CLUS-HSC, both for tree and DAG structured class hierarchies, and for all evaluation measures. Somewhat unexpectedly, learning a single-label tree for each class separately, where one only focuses on the examples belonging to the parent class, results in lower predictive performance than learning one single model for all classes. That is, CLUS-HMC outperforms CLUS-HSC. CLUS-HSC in turn outperforms CLUS-SC for DAGs; for trees the performances are similar.

Second, we have compared the precision-recall behavior of the algorithms to that of a default model. Using micro-averaged PR curves, we have observed that CLUS-SC performs consistently (for 23 out of 24 datasets) worse than default, indicating that it builds overly complex models that overfit the training data. Interestingly, the other precision-recall averaging methods are not able to detect this overfitting.

Third, the size of the HMC tree is much smaller (2 to 3 orders of magnitude) than the total size of the models output by CLUS-HSC and CLUS-SC. As was expected, the CLUS-HSC models are smaller than the CLUS-SC models (a factor 2 to 3).

Fourth, we find that learning a single HMC tree is also much faster than learning many regular trees. Not only is CLUS-HMC more efficient that CLUS-HSC, it turns out to be also more efficient than CLUS-HSC. Obviously, a single HMC tree is also much more efficient to apply than 4000 (for GO) separate trees.

Given the positive results for HMC decision trees on predictive performance, model size, and efficiency, we can conclude that their use should definitely be considered in HMC tasks where interpretable models are desired.

# Chapter 4

# Decision Tree Ensembles for Gene Function Prediction

## 4.1 Introduction

In the previous chapter, we have shown that HMC trees, which predict all the classes of the hierarchy in one single model, are to be preferred for HMC tasks where an interpretable model is desired. Therefore, we have studied the properties of three decision tree approaches towards HMC from a machine learning viewpoint. In this chapter, we will turn to the specific application area of functional genomics: we will present ensembles of HMC trees and compare HMC trees as well as their ensembles to state-of-the-art approaches found in the biomedical literature that were specifically developed for this task.

*S. cerevisiae* (yeast), *A. thaliana* (thale cress) and *M. musculus* (mouse) are well-studied organisms in biology, and the sequencing of their genomes was completed many years ago. It is still a challenge, however, to develop methods that assign biological functions to the open reading frames (ORFs) in these genomes automatically. Different machine learning methods have been proposed to this end. These approaches differ with respect to a number of characteristics: which learning algorithm they are based on, whether the hierarchy constraint is always met and whether they can deal with hierarchies structured as a directed acyclic graph (DAG), such as the Gene Ontology, or are restricted to hierarchies structured as a rooted tree, like MIPS's FunCat. In general, it remains unclear which method is to be preferred in terms of predictive performance, efficiency, interpretability, and usability.

The motivation for using decision trees in this context is the following: they are a well-known type of classifiers that can be learned efficiently from large datasets, produce accurate predictions and can lead to knowledge that provides insight in

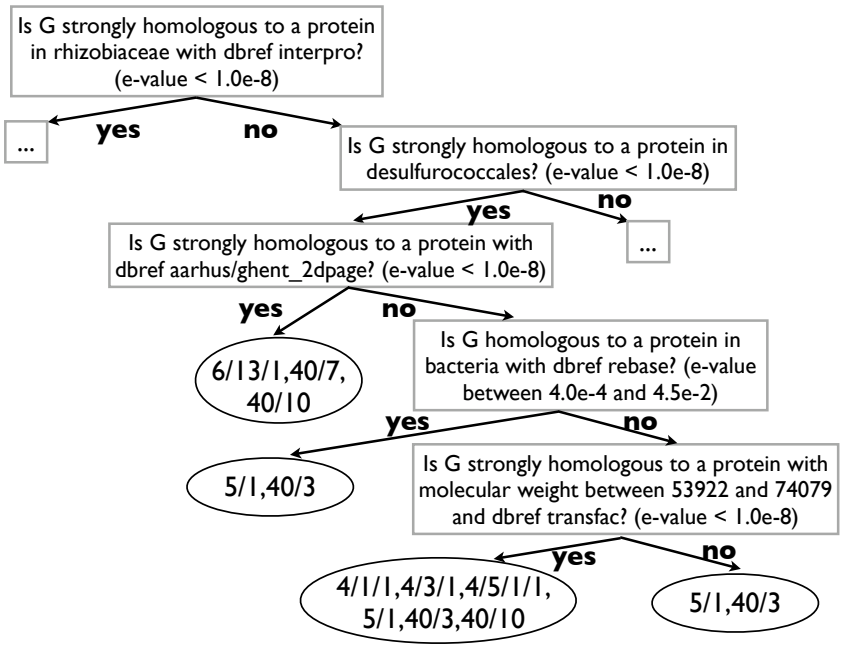


Figure 4.1: Example of a predictive clustering tree. This tree predicts the functions of a gene $G$, based on homology data. The functions are taken from the FunCat classification scheme [Mewes et al., 1999] and are hierarchical: if for example function 4/3/1 (tRNA synthesis) is predicted, then function 4/3 (tRNA transcription) and function 4 (transcription) are predicted as well.

the biology behind the predictions, as demonstrated by Clare and King [2003]. Figure 4.1 gives an example of a (partial) predictive clustering tree predicting the functions of *S. cerevisiae* genes from homology data [Clare, 2003]. The homology features are based on a sequence similarity search performed for each yeast gene against all the genes in SwissProt. Each internal node of the tree contains a test on one of the attributes in the dataset. Here, the attributes are binary and have been obtained after preprocessing the relational homology data with a frequent pattern miner. The root node, for instance, tests whether there exists a SwissProt protein that has a high similarity (e-value $< 1.0 \cdot 10^{-8}$) with the gene under consideration $G$, is classified into the rhizobiaceae group and has references to the InterPro database. In order to predict the functions of a new gene, the gene is routed down the tree according to the outcome of the tests. When a leaf node is reached, the gene is assigned the functions that are stored in it. Only the most specific functions are shown in the figure. Recall from Sect. 3.3.3 that $\bar{v}_i$

is the proportion of examples in the leaf belonging to $c_i$. An example arriving in the leaf can therefore be predicted to belong to class $c_i$ if $\bar{v}_i$ is above some threshold $t_i$, which can be chosen by the user. To ensure that the predictions obey the hierarchy constraint (whenever a class is predicted its superclasses are also predicted), it suffices to choose $t_i \leq t_j$ whenever $c_i$ is a superclass of $c_j$. The PCT in Fig. 4.1 has a threshold of $t_i = 0.4$ for all $i$.

The contributions of this chapter are twofold:

- We show that CLUS-HMC outperforms previously published approaches applied to *S. cerevisiae* and *A. thaliana* [Clare, 2003; Clare et al., 2006].
- We show that by upgrading our method to an ensemble technique, called CLUS-HMC-ENS, classification performance improves further. Ensemble techniques are learning methods that construct a set of classifiers and classify new data instances by taking a vote over their predictions. Experiments show that CLUS-HMC-ENS outperforms Bayesian corrected support vector machines [Barutcuoglu et al., 2006], a statistical learning method for gene function prediction, on *S. cerevisiae* data, and methods participating in the MouseFunc challenge [Hughes and Roth, 2008; Pena-Castillo et al., 2008] on *M. musculus* data.

The contents of this chapter are the result of joint work with Celine Vens, Jan Struyf and Hendrik Blockeel at the Katholieke Universiteit Leuven and Dragi Kocev and Sašo Džeroski at the Jožef Stefan Institute Ljubljana. They have been published in [Blockeel et al., 2006; Schietgat et al., 2010].

The chapter is organised as follows. We start by discussing previous work in Sect. 4.2. Section 4.3 presents the upgrade of CLUS-HMC to its ensemble version CLUS-HMC-ENS. In Sect. 4.4, we give experiments and in Sect. 4.5 we conclude.

## 4.2 Related work

In Sect. 3.2, related work on hierarchical multi-label classification was discussed within the general machine learning domain. Here, we specifically discuss related work from the biomedical literature. A number of machine learning approaches have been proposed in the area of functional genomics. They have been applied in the context of gene function prediction in *S. cerevisiae*, *A. thaliana* or *M. musculus*. We have grouped them according to the learning approach they use.

### 4.2.1 Network based methods

Several approaches predict functions of unannotated genes based on known functions of genes that are nearby in a functional association network or protein-protein interaction network [Troyanskaya et al., 2003; Karaoz et al., 2004; Chua

et al., 2006]. GeneFAS [Chen and Xu, 2004], for example, predicts functions of unannotated yeast genes based on known functions of genes that are nearby in a functional association network. GeneMANIA [Mostafavi et al., 2008] calculates per gene function a composite functional association network from multiple networks derived from different genomic and proteomic data sources.

These approaches are based on label propagation and do not return a global predictive model. However, a number of approaches were proposed to combine predictions of functional networks with those of a predictive model. Kim et al. [2008] combine them with predictions from a Naive Bayes classifier. The combination is based on a simple aggregation function. The Funckenstein system [Tian et al., 2008] uses logistic regression to combine predictions made by a functional association network with predictions from a random forest.

### 4.2.2 Kernel based methods

Deng et al. [2002] predict gene functions with Markov random fields using protein interaction data. They learn a model for each gene function separately and ignore the hierarchical relationships between the functions. Lanckriet et al. [2004] represent the data by means of a kernel function and construct support vector machines for each gene function separately. They only predict top-level classes in the hierarchy. Lee et al. [2006] have combined the Markov random field approach of Deng et al. [2002] with the SVM approach of Lanckriet et al. [2004] by computing diffusion kernels and using them in kernel logistic regression.

Obozinski et al. [2008] present a two-step approach in which SVMs are first learned independently for each gene function separately (allowing violations of the hierarchy constraint) and are then reconciled to enforce the hierarchy constraint. Barutcuoglu et al. [2006] have proposed a similar approach where unthresholded support vector machines are learned for each gene function and then combined using a Bayesian network so that the predictions are consistent with the hierarchical relationships. Guan et al. [2008] extend this method to an ensemble framework that is based on three classifiers: a classifier that learns a single support vector machine for each gene function, the Bayesian corrected combination of support vector machines mentioned above, and a classifier that constructs a single support vector machine per gene function and per data source and forms a Naive Bayes combination over the data sources.

Methods that learn a separate model for each function have several disadvantages (see Sect. 3.3.6). Firstly, they are usually less efficient, because $n$ models have to be built (with $n$ the number of functions). Secondly, they often learn from strongly skewed class distributions, which is difficult for many learners.

The disadvantage of subsymbolic learning techniques, such as support vector machines, is the lack of interpretability: it is very hard to find out why a support vector machine assigns certain classes to an example, especially if a non-linear kernel is used. In contrast to the output of kernel based methods, decision trees,

which we discuss next, usually lend themselves to interpretation by a domain expert.

### 4.2.3 Decision tree based methods

Clare [2003] presents an HMC decision tree method that learns a single tree for predicting gene functions of *S. cerevisiae*. She adapts the well-known decision tree algorithm C4.5 [Quinlan, 1993] to cope with the issues introduced by the HMC task. First, where C4.5 normally uses class entropy for choosing the best split, her version uses the sum of entropies of the class variables. Second, she extends the method to predict classes on several levels of the hierarchy, assigning a larger cost to misclassifications higher up in the hierarchy. A fixed classification threshold of 0.5 is chosen. The resulting tree is transformed into a set of rules, and the best rules are selected, based on a significance test performed on a separate validation set. Note that this last step violates the hierarchy constraint, since rules predicting a class can be dropped while rules predicting its subclasses are kept. The non-hierarchical version of her method was later used to predict GO terms for *A. thaliana* [Clare et al., 2006]. Here, the annotations are predicted for each level of the hierarchy separately.

Hayete and Bienkowska [2005] build a decision tree for each GO function separately using information about protein assignments in the same functional domain. As mentioned in Sect. 3.3.6, methods that learn separate models for each function have several disadvantages.

## 4.3 Ensembles of HMC decision trees

In this section we will show how Clus-HMC, the HMC decision tree approach introduced in the previous chapter, can be extended to an ensemble version.

Ensemble methods are learning methods that construct a set of classifiers for a given prediction task and classify new examples by combining the predictions of each classifier. Here we consider bagging, an ensemble learning technique that has primarily been used in the context of decision trees. In preliminary experiments, we also considered two other ensemble learning techniques: random forests [Breiman, 2001] and an adapted version of the boosting approach for regression trees by Drucker [1997]. However, neither method performed better than simple bagging.

Bagging [Breiman, 1996a] is an ensemble method where the different classifiers are constructed by making bootstrap replicates of the training set and using each of these replicates to construct one classifier. Each bootstrap sample is obtained by randomly sampling training instances, with replacement, from the original training set, until the sample contains the same number of instances as the original training set. The individual predictions given by each classifier can be combined by taking the average (for numeric targets) or the majority vote (for nominal targets).

**Algorithm 4.1** CLUS-HMC-ENS: bagging algorithm using the PCT algorithm as base classifier.

**procedure** bagging($I, k$) **returns** forest
1: **for** $i = 1$ **to** $k$
2: $I_i$ = sample_with_replacement($I$)
3: $T_i$ = PCT($I_i$)
4: **return** $\bigcup_i \{T_i\}$

The algorithm for bagging the PCTs is given in Table 4.1. It takes an extra parameter $k$ as input that denotes the number of trees in the ensemble. In order to make predictions, the average of all class vectors predicted by the $k$ trees in the ensemble is computed, and then the threshold is applied as before. This ensures that the hierarchy constraint holds. We call the resulting instantiation of the bagging algorithm around the CLUS-HMC algorithm CLUS-HMC-ENS.

Breiman has shown that bagging can give substantial gains in the predictive performance of decision tree learners [Breiman, 1996a]. Also in the case of learning PCTs for predicting multiple targets at once (multi-task learning [Caruana, 1997]), decision tree methods benefit from the application of bagging [Kocev et al., 2007].

By using bagging on top of the PCT algorithm, we decrease the interpretability of the resulting model: instead of one tree, we obtain a collection of trees whose predictions are averaged. Thus, there is a clear trade-off between predictive performance and interpretability to be considered by the user. However, methods exist that overcome the comprehensibility issue of ensembles. For example, Van Assche and Blockeel [2007] learn a single decision tree that approximates an ensemble of trees. The accuracy of the resulting tree is significantly better than that of a single tree learned directly from the data (and on average close to that of bagging). Also, feature rankings that explain the relative importance of the individual explanatory variables can be obtained from an ensemble of trees [Breiman, 2001].

## 4.4 Experiments

In this section, we address the following questions:

- What is the improvement in predictive performance, if any, that can be obtained by using CLUS-HMC-ENS on functional genomics data?
- How does the predictive performance of CLUS-HMC and CLUS-HMC-ENS compare to results reported in the biomedical literature?

In order to answer these questions, we compare our results to the results reported by Clare and King [2003] and Barutcuoglu et al. [2006] on *S. cerevisiae*, to the

results reported by Clare et al. [2006] on *A. thaliana*, and to the results of the groups participating in the MouseFunc challenge [Hughes and Roth, 2008; Pena-Castillo et al., 2008] on *M. musculus*.

### 4.4.1 Datasets

For *S. cerevisiae* and *A. thaliana*, the datasets that we use in our evaluation are exactly those datasets that are used in the cited articles. They are available, together with the parameter settings that can be used to reproduce the results, at the following webpage: `http://www.cs.kuleuven.be/~dtai/clus/hmc-ens`. For *M. musculus*, the (raw) data is available at `http://hugheslab.med.utoronto.ca/supplementary-data/mouseFunc_I/`, while the dataset we assembled from it is available at the former webpage.

Next to predicting gene functions of three organisms (*S. cerevisiae*, *A. thaliana*, and *M. musculus*), we consider two annotation schemes in our evaluation: FunCat, which is a tree-structured class hierarchy and the Gene Ontology, which forms a directed acyclic graph instead of a tree: each term can have multiple parents.

#### 4.4.1.1 Saccharomyces cerevisiae

The first dataset we use ($\mathbf{D_0}$) was described by Barutcuoglu et al. [2006] and is a combination of different data sources. The input feature vector for a gene consists of pairwise interaction information, membership to colocalisation locale, possession of transcription factor binding sites and results from microarray experiments, yielding a dataset with in total 5930 features. The 3465 genes are annotated with function terms from a subset of 105 nodes from the Gene Ontology's *biological process* hierarchy.

We also use the 12 yeast datasets ($\mathbf{D_1} - \mathbf{D_{12}}$) from Clare [2003] that were listed in Table 3.2. However, instead of using the updated class labels as in the previous chapter, we will use the original labels of the datasets. The reason for this is to make a comparison with Clare [2003] possible. Only annotations from the first four levels are given there. The description of these datasets can be found in Sect. 3.6.1.

#### 4.4.1.2 Arabidopsis thaliana

We use six datasets from Clare et al. [2006], originating from different sources: sequence statistics, expression, predicted SCOP class, predicted secondary structure, InterPro and homology. Each dataset comes in two versions: with annotations from the FunCat classification scheme and from the Gene Ontology's *molecular function* hierarchy. Again, only annotations for the first four levels are given. We use the manual annotations for both schemes.

**$D_{13}$** (seq) records sequence statistics in exactly the same way as for *S. cerevisiae*. **$D_{14}$** (exprindiv) contains 43 experiments from NASC's Affymetrix service "Affywatch" (`http://affymetrix.arabidopsis.info/AffyWatch.html`), taking the signal, detection call and detection $p$-values. **$D_{15}$** (scop) consists of SCOP superfamily class predictions made by the Superfamily server [Gough et al., 2001]. **$D_{16}$** (struc) was obtained in the same way as for *S. cerevisiae*. **$D_{17}$** (interpro) includes features from several motif or signature finding databases, like PROSITE, PRINTS, Pfam, ProDom, SMART and TIGRFAMs, calculated using the EBI's stand-alone InterProScan package [Zdobnov and Apweiler, 2001]. To obtain features, the relational data was mined in the same manner as the structure data. **$D_{18}$** (hom) was obtained in the same way as for *S. cerevisiae*, but now using SwissProt v41.

#### 4.4.1.3 Mus musculus

We use the data that was provided for the MouseFunc challenge [Hughes and Roth, 2008; Pena-Castillo et al., 2008]. It consists of 21603 genes, of which 1718 are set aside as test genes. Each gene is annotated with GO terms from a specified subset of the Gene Ontology. The annotations are up-propagated using the Gene Ontology's "is-a" and "part-of" relation. The data is composed of several sources: gene expression data, protein sequence pattern annotations, protein-protein interactions, phenotype annotations, phylogenetic profile and disease associations. In order to construct a single dataset (**$D_{19}$**), we joined all data tables, removed attributes with fewer than five non-zero values and performed a naive propositionalisation of protein interaction information by computing additional attributes that indicate for each gene the classes of other genes to which it is linked through a protein-protein interaction (only considering training set genes). This yields 18746 attributes in total. The resulting representation is similar to the one used by Guan et al. [2008].

### 4.4.2 Method

**Evaluation measures** As in the previous chapter, we report the performance of the different methods with precision-recall (PR) based evaluation measures. This is motivated by the following three observations: (1) precision-recall evaluation was used in earlier approaches to gene function prediction [Deng et al., 2002; Chua et al., 2006; Pena-Castillo et al., 2008], (2) it allows one to simultaneously compare classifiers for different classification thresholds, and (3) it suits the characteristics of typical HMC datasets, in which many classes are infrequent (i.e., typically only a few genes have a particular function). Viewed as a binary classification task for each class, this implies that for most classes the number of negative instances by far exceeds the number of positive instances. In some cases, it is preferred to recognise the positive instances (i.e., that a gene has a given function), rather than correctly

predicting the negative ones (i.e., that a gene does not have a particular function). ROC curves [Provost and Fawcett, 1998] are then less suited for this task, exactly because they reward a learner if it correctly predicts negative instances (giving rise to a low false positive rate). This can present an overly optimistic view of the algorithm's performance [Davis and Goadrich, 2006].

The definition of precision, recall, average precision and average recall can be found in Sect. 3.5. Note that these measures ignore the number of correctly predicted negative examples. A precision-recall curve (PR curve) plots the precision of a model as a function of its recall. Here, we consider two types of PR curves: (1) the average or pooled PR curve, which plots $\overline{Precision}$ versus $\overline{Recall}$ and summarises the performance of the model across all functions, and (2) the function-specific PR curve for a given function $i$, which plots $Precision_i$ versus $Recall_i$. Such curves allow us to evaluate the predictive performance of a model regardless of $t$. In the end, a domain expert can choose the threshold corresponding to the point on the curve that looks most interesting to him.

Although a PR curve helps in understanding the predictive behaviour of the model, a single performance score is more useful to compare models. Therefore, we use the two AUPRC measures that correspond to the two types of PR curves and that are often reported in the literature: AU($\overline{\text{PRC}}$), the area under the average PR curve (defined in Sect. 3.5.2.1), and $\overline{\text{AUPRC}}$, the average over all areas under the function-specific PR curves (defined in Sect. 3.5.2.2). Note that AU($\overline{\text{PRC}}$) gives more weight to more frequent functions, while $\overline{\text{AUPRC}}$ considers the importance of every function to be equal.

**Parameter settings for Clus-HMC and Clus-HMC-Ens** In the experiments, $w_0$, which determines the weights of the different functions in the decision tree heuristic, is set to 0.75 and the number of examples in each decision tree leaf is lower bounded to 5. The parameter $k$, which denotes the number of trees used in the ensemble, is set to 50. Preliminary experiments show that performance does not strongly depend on the choice of $w_0$ and that it does not significantly increase after $k = 50$, so the latter value is a good trade-off between performance and runtime. The significance parameter used in the $F$-test stopping criterion of Clus-HMC and Clus-HMC-Ens is tuned on a separate validation set (1/3 of the training data) and optimised out of 6 possible values (0.001, 0.005, 0.01, 0.05, 0.1, 0.125), maximising the AU($\overline{\text{PRC}}$). The final model is constructed on the entire training set using the selected value of the significance parameter.

### 4.4.3 Results

We will first investigate if ensembles improve the predictive performance of Clus-HMC in gene function prediction and if so, quantify this difference. We will then compare Clus-HMC and Clus-HMC-Ens against several state-of-the-art

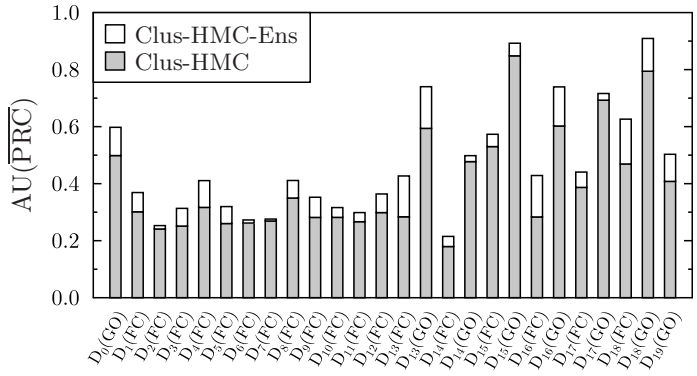


Figure 4.2: Comparison of AU($\overline{\text{PRC}}$) between Clus-HMC and Clus-HMC-Ens. The white surface represents the gain in AU($\overline{\text{PRC}}$) obtained by Clus-HMC-Ens.

systems in gene function prediction. On the one hand, we will compare Clus-HMC to C4.5H/M [Clare and King, 2003; Clare et al., 2006], because they both build a single decision tree. On the other hand, we will compare Clus-HMC-Ens to Bayesian-corrected SVMs [Barutcuoglu et al., 2006], a statistical learning approach, on $D_0$, and to the methods that entered the MouseFunc challenge on $D_{19}$.

The datasets originating from Clare and King [2003] and Clare et al. [2006] (i.e., datasets $D_1$ to $D_{18}$) are divided into a training set (2/3) and a test set (1/3). We use exactly the same splits. For dataset $D_0$, we randomly construct a training and test set with the same ratio. For dataset $D_{19}$, we use the same training and test sets that were used in the MouseFunc challenge.

#### 4.4.3.1 Comparison between Clus-HMC and Clus-HMC-Ens

For each of the datasets, the AU($\overline{\text{PRC}}$) of Clus-HMC and Clus-HMC-Ens is shown in Fig. 4.2. We see that for every dataset, there is an increase in AU($\overline{\text{PRC}}$) when using ensembles. The average gain is 0.071 (which is an improvement of 18% on average); the maximal gain is 0.157. Representative PR curves can be found in Fig. 4.3.

Figure 4.4 shows the $\overline{\text{AUPRC}}$ of Clus-HMC and Clus-HMC-Ens. Again, there is an increase in $\overline{\text{AUPRC}}$ when using ensembles, with an average gain of 0.093 (which is an improvement of 108% on average) and a maximal gain of 0.337. These results show that the increase in performance obtained by Clus-HMC-Ens is larger according to $\overline{\text{AUPRC}}$ than according to AU($\overline{\text{PRC}}$), which indicates that ensembles are performing particularly better for the less frequent classes.

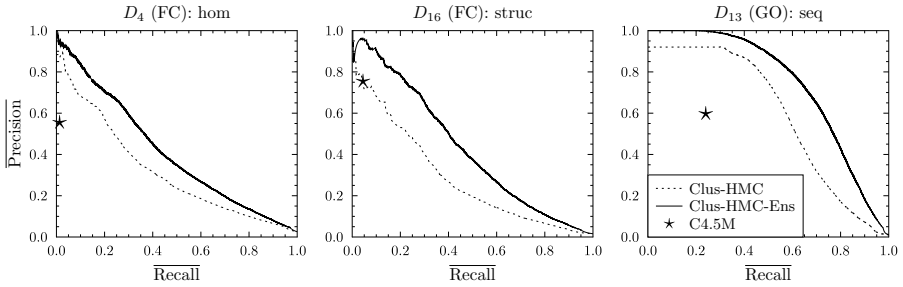


Figure 4.3: Average PR curves for Clus-HMC, Clus-HMC-Ens and C4.5H/M on $D_4$ and $D_{16}$ with FunCat annotations and on $D_{13}$ with GO annotations.

**Conclusion** The improvement in predictive performance that can be obtained by using tree ensembles in more straightforward machine learning settings carries over to the HMC setting with functional genomics data.

#### 4.4.3.2 Comparison between Clus-HMC and C4.5H/M

We now concentrate on the comparison of the results obtained by our algorithms to those obtained by other decision tree based algorithms. For the datasets that are annotated with FunCat classes ($D_1 - D_{18}$), we will compare to the hierarchical extension of C4.5 [Clare and King, 2003], which we will refer to as C4.5H. For the datasets with GO annotations ($D_{13} - D_{18}$), we will use the non-hierarchical multi-label extension of C4.5 [Clare et al., 2006], as C4.5H cannot handle hierarchies structured as a DAG. We refer to this system as C4.5M.

For their experiments on *A. thaliana*, Clare et al. [2006] only report results per level of the hierarchy. In order to obtain these results, they learn a separate classifier per level, removing from their training and test set those genes that do not have annotated functions at that level. This approach may give a biased result: when annotating a new gene, it is not known in advance at which levels of the hierarchy it will have functions. Therefore, we reran C4.5M to learn one classifier that uses all training data and tested it on the complete test set.

For evaluating their systems, Clare and King [2003] and Clare et al. [2006] report precision. Indeed, as the biological experiments required to validate the learned rules are costly, it is important to avoid false positives. However, precision is always traded off by recall: a classifier that predicts one example positive, but misses 1000 other positive examples may have a precision of 1, although it can hardly be called a good classifier. Therefore, we also compute the recall of the models obtained by C4.5H/M. These models were presented as rules for specific classes without any probability scores, so each model corresponds to precisely one

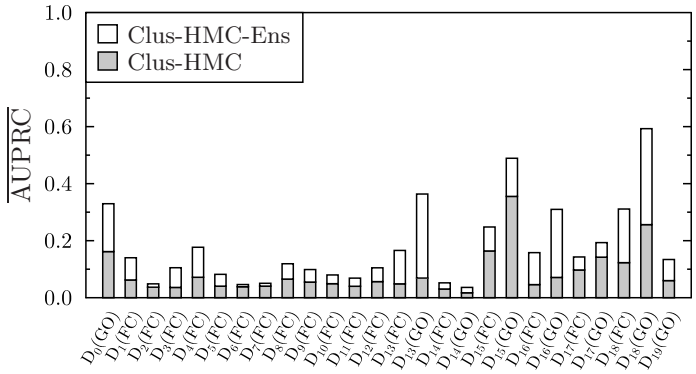


Figure 4.4: Comparison of $\overline{\text{AUPRC}}$ between Clus-HMC and Clus-HMC-Ens. The white surface represents the gain in $\overline{\text{AUPRC}}$ obtained by Clus-HMC-Ens.

point in PR space.

For each of the datasets $D_1 - D_{18}$, these PR points are plotted against the average PR curves for Clus-HMC. As we are comparing curves with points, we speak of a "win" for Clus-HMC when its curve is above C4.5H/M's point, and of a "loss" when it is below the point. Under the null hypothesis that both systems perform equally well, we expect as many wins as losses. We observed that only in one case out of 24, for dataset $D_{16}$ with FunCat annotations, C4.5H/M outperforms Clus-HMC. For all other cases there is a clear win for Clus-HMC. Representative PR curves can be found in Fig. 4.3.

For each of these datasets, we also compared the precision of C4.5H/M, Clus-HMC and Clus-HMC-Ens, at the recall obtained by C4.5H/M. The results can be found in Fig. 4.5. The average gain in precision w.r.t. C4.5H/M is 0.209 for Clus-HMC and 0.276 for Clus-HMC-Ens.

**Conclusion** Clus-HMC is the tree-building system that yields the best predictive performance. Compared with other existing methods, we are able to obtain the same precision with higher recall, or the same recall with higher precision. An explanation for this difference could lie in the selection of a fixed classification threshold ($t = 0.5$) by C4.5H/M. Moreover, the hierarchy constraint is always fulfilled, which is not the case for C4.5H/M.

**Comparing individual rules** Every leaf of a decision tree corresponds to an *if ... then ...* rule. When comparing the complexity and precision/recall of these individual rules, Clus-HMC also performs well. For instance, take FunCat class 29, which has a prior frequency of 3%. Figure 4.6 shows the PR evaluation for

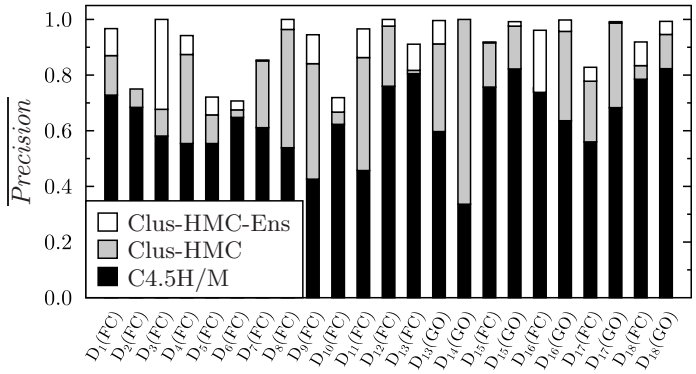


Figure 4.5: Comparison of precision between C4.5H/M, Clus-HMC and Clus-HMC-Ens, at the recall obtained by C4.5H/M. The gray surface represents the gain in precision obtained by Clus-HMC, the white surface represents the gain for Clus-HMC-Ens. $D_{14}$(FC) was not included, since C4.5H did not find significant rules. For $D_{16}$(FC), C4.5H scored a slightly better precision (see Fig. 4.3), hence the lack of gray surface.

the algorithms for this class using homology dataset $D_4$. The PR point for C4.5H corresponds to one rule, shown in Fig. 4.7. This rule has a precision/recall of 0.55/0.17. Clus-HMC's most precise rule for class 29 is shown in Fig. 4.8. This rule has a precision/recall of 0.90/0.26.

Note from Fig. 4.6 that an even higher precision can be obtained with Clus-HMC-Ens, although the rules which lead to this prediction are more complex.

#### 4.4.3.3 Comparison between Clus-HMC-Ens and Bayesian-corrected SVMs

In this section, we compare Clus-HMC-Ens to the statistical learning method of Barutcuoglu et al. [2006], which consists of Bayesian-corrected SVMs (see Sect. 4.2). We will further refer to this method as BSVM. The authors have used dataset $D_0$ to evaluate their method and report class-specific area under the ROC convex hull (AUROC) for a small subset of 105 nodes of the Gene Ontology. As only AUROC scores are reported by Barutcuoglu et al. [2006], we adopt the same evaluation metric for this comparison.

Barutcuoglu et al. [2006] build a bagging procedure around their system and report out-of-bag error estimates [Breiman, 1996b] as evaluation, which removes the need for a set-aside test set. Out-of-bag error estimation proceeds as follows: for each example in the original training set, the predictions are made by aggregating only over those classifiers for which the example was not used for training.

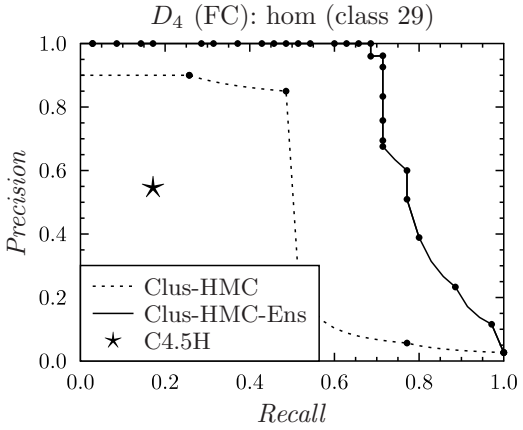


Figure 4.6: Precision-recall curve for class 29 on $D_4$ with FunCat annotations.

| | |
|---|---|
| **if** | the ORF is NOT homologous to another yeast protein ($e \geq 0.73$) and homologous to a protein in rhodospirillaceae ($e < 1.0 \cdot 10^{-8}$) |
| **and** | NOT homologous to another yeast protein ($5.0 \cdot 10^{-4} < e < 3.3 \cdot 10^{-2}$) and homologous to a protein in anabaena ($e \geq 1.1$) |
| **and** | homologous to another yeast protein ($2.0 \cdot 10^{-7} < e < 5.0 \cdot 10^{-4}$) and homologous to a protein in beta_subdivision ($e < 1.0 \cdot 10^{-8}$) |
| **and** | NOT homologous to a protein in sinorhizobium with keyword transmembrane ($e \geq 1.1$) |
| **and** | NOT homologous to a protein in entomopoxvirinae with dbref pir ($e \geq 1.1$) |
| **and** | NOT homologous to a protein in t4-like_phages with molecular weight between 1485 and 38502 ($4.5 \cdot 10^{-2} < e < 1.1$) |
| **and** | NOT homologous to a protein in chroococcales with dbref prints ($1.0 \cdot 10^{-8} < e < 4.0 \cdot 10^{-4}$) |
| **and** | NOT homologous to a protein with sequence length between 344 and 483 and dbref tigr ($e < 1.0 \cdot 10^{-8}$) |
| **and** | homologous to a protein in beta_subdivision with sequence length between 16 and 344 ($e < 1.0 \cdot 10^{-8}$) |
| **then** | class 29/0/0/0 "transposable elements, viral and plasmid proteins" |

Figure 4.7: Rule found by C4.5H on the $D_4$(FC) homology dataset, with a precision of 0.55 and a recall of 0.17.

| | |
|---|---|
| **if** | the ORF is NOT homologous to a protein in rhizobiaceae_group with dbref interpro ($e < 1.0 \cdot 10^{-8}$) |
| **and** | NOT homologous to a protein in desulfurococcales ($e < 1.0 \cdot 10^{-8}$) |
| **and** | homologous to a protein in ascomycota with dbref transfac ($e < 1.0 \cdot 10^{-8}$) |
| **and** | homologous to a protein in viridiplantae with sequence length $\geq 970$ ($e < 1.0 \cdot 10^{-8}$) |
| **and** | homologous to a protein in rhizobium with keyword plasmid ($1.0 \cdot 10^{-8} < e < 4.0 \cdot 10^{-4}$) |
| **and** | homologous to a protein in nicotiana with dbref interpro ($e < 1.0 \cdot 10^{-8}$) |
| **then** | class 29/0/0/0 "transposable elements, viral and plasmid proteins" |

Figure 4.8: Rule found by Clus-HMC on the $D_4$(FC) homology dataset, with a precision of 0.90 and a recall of 0.26.

This is the out-of-bag classifier. The out-of-bag error estimate is then the error rate of the out-of-bag classifier on the training set. The number of bags used in this procedure was 10. To compare our results, we use exactly the same method.

On dataset $D_0$, the average of the AUROC over the 105 functions is 0.871 for Clus-HMC-Ens and 0.854 for BSVM. Figure 4.9 compares the class-specific out-of-bag AUROC estimates for Clus-HMC-Ens and BSVM outputs. Clus-HMC-Ens scores better on 73 of the 105 functions, while BSVM scores better on the remaining 32 cases. According to the (two-sided) Wilcoxon signed rank test [Wilcoxon, 1945], the performance of Clus-HMC-Ens is significantly better ($p = 4.37 \cdot 10^{-5}$).

Moreover, Clus-HMC-Ens is faster than BSVM. Runtimes are compared for one of the datasets having annotations from Gene Ontology's complete *biological process* hierarchy (in particular, we used $D_{16}$, which is annotated with 629 classes). Run on a cluster of AMD Opteron processors (1.8 - 2.4GHz, $\geq$2GB RAM), Clus-HMC-Ens required 15.9 hours, while SVM-light [Joachims, 1999], which is the first step of BSVM, required 190.5 hours for learning the models (i.e., Clus-HMC-Ens is faster by a factor 12 in this case).

**Conclusion** On dataset $D_0$, Clus-HMC-Ens has a significantly better predictive performance (measured in AUROC), while it is faster to run.

#### 4.4.3.4 Comparison between Clus-HMC-Ens and the methods in the MouseFunc challenge

In this section we compare Clus-HMC-Ens to the seven systems that submitted predictions to the MouseFunc challenge. These systems are the ensemble exten-

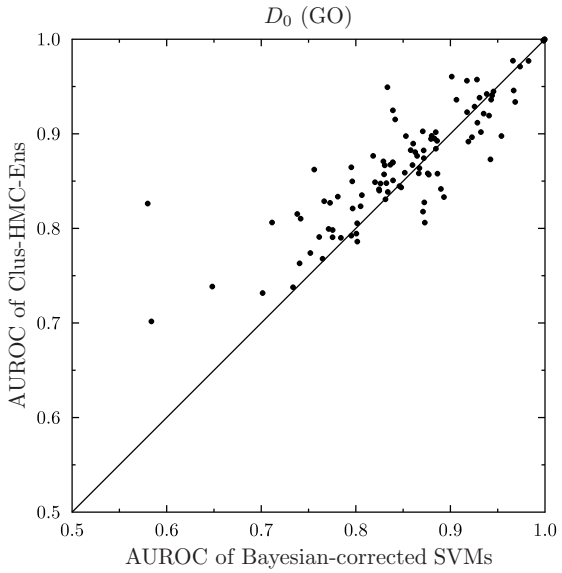


Figure 4.9: Class-specific out-of-bag AUROC comparison between CLUS-HMC-ENS and Bayesian-corrected SVMs.

sion of BSVM [Guan et al., 2008] (which we will call BSVM$^+$), Kernel Logistic Regression [Lee et al., 2006] (which we will call KLR), calibrated SVMs [Obozinski et al., 2008] (which we will call CSVM), GENEFAS [Chen and Xu, 2004], GENEMANIA [Mostafavi et al., 2008], the combined functional network and classifier strategy of Kim et al. [2008] (which we will call KIM) and the Funckenstein system [Tian et al., 2008]. These methods were described in Sect. 4.2. Note that, when comparing the results, one should keep in mind that each team independently constructed a dataset, possibly using different features. As a result, the differences in performance can be due not only to the learning methods compared, but also the different feature sets used by the methods. As mentioned in Sect. 4.4.1.3, the representation that we use is the one of the BSVM$^+$ team.

Each method gives predictions for 2815 selected GO terms.[1] Because the MouseFunc website lists a prediction matrix (containing for each gene-term pair the corresponding probability that the gene is annotated with the GO term) for each of the methods we compare to, we can easily compute the PR based evaluation measures on these predictions, using the Clus sofware.

[1]One GO term (GO:0005489) was declared obsolete in the GO version used in the challenge. For this reason, in one of the 12 subsets ("MF_31-100"), we replaced all annotations with this term by the new term (GO:0009055), as indicated in the GO hierarchy file.

The 2815 GO terms are divided into 12 disjunct subsets corresponding to all combinations of the three GO branches (Biological Process, Molecular Function and Cellular Component) with four ranges of specificity, which is defined as the number of genes in the training set to which each term is associated (3-10, 11-30, 31-100 and 101-300). We have adopted the same subsets and trained and evaluated our models on each of them. Since 1846 of the selected 2815 GO terms were used as annotation in the test set, our evaluation of all the systems is based only on those.

Table 4.1 shows the AU($\overline{\text{PRC}}$) results of all the methods on the 12 subsets. Looking at the wins/losses for each of the 12 subsets, according to the (two-sided) Wilcoxon signed rank test, the performance of Clus-HMC-Ens is significantly better at the 1% level than BSVM$^+$ ($p = 4.88 \cdot 10^{-4}$), CSVM ($p = 1.47 \cdot 10^{-3}$), GeneFAS ($p = 4.88 \cdot 10^{-4}$) and KIM ($p = 4.88 \cdot 10^{-4}$). Clus-HMC-Ens has more wins than KLR ($p = 1.61 \cdot 10^{-2}$) and GeneMANIA ($p = 1.61 \cdot 10^{-2}$), but is not significantly better at 1%. Clus-HMC-Ens is performing significantly worse than Funckenstein ($p = 9.28 \cdot 10^{-3}$).

Table 4.2 shows the same comparison, but now for $\overline{\text{AUPRC}}$. According to the Wilcoxon signed rank test, Clus-HMC-Ens is performing significantly better at the 1% level than KIM ($p = 4.88 \cdot 10^{-4}$), while it is not significantly different from BSVM$^+$ ($p = 4.70 \cdot 10^{-1}$), KLR ($p = 1.61 \cdot 10^{-2}$), CSVM ($p = 1.51 \cdot 10^{-1}$) and GeneFAS ($p = 2.59 \cdot 10^{-2}$). Clus-HMC-Ens is performing significantly worse than GeneMANIA ($p = 9.28 \cdot 10^{-3}$) and Funckenstein ($p = 9.77 \cdot 10^{-4}$).

Because $\overline{\text{AUROC}}$, the average over all areas under the function-specific ROC curves, was used as evaluation measure in the MouseFunc challenge [Pena-Castillo et al., 2008], we report it in Table 4.3. According to the Wilcoxon signed rank test, Clus-HMC-Ens is not performing significantly different at the 1% level than KLR ($p = 9.10 \cdot 10^{-1}$), CSVM ($p = 2.20 \cdot 10^{-2}$), GeneFAS ($p = 5.69 \cdot 10^{-1}$) and KIM ($p = 3.22 \cdot 10^{-2}$). Clus-HMC-Ens is performing significantly worse than BSVM$^+$ ($p = 4.88 \cdot 10^{-4}$), GeneMANIA ($p = 9.77 \cdot 10^{-4}$) and Funckenstein ($p = 9.77 \cdot 10^{-4}$).

The fact that Clus-HMC-Ens performs better according to AU($\overline{\text{PRC}}$) than to $\overline{\text{AUPRC}}$ and $\overline{\text{AUROC}}$ can be explained as follows. The variance function used to select the best tests gives a higher weight to functions at higher levels of the hierarchy, causing Clus-HMC-Ens to perform well especially on those functions. In contrast to $\overline{\text{AUPRC}}$ and $\overline{\text{AUROC}}$, which consider each function as equal, the AU($\overline{\text{PRC}}$) evaluation measure shares the idea of giving a higher penalty to mistakes made for functions at higher levels of the hierarchy.

**Conclusion** In general, the performance of Clus-HMC-Ens is not significantly different from that of BSVM$^+$, which has been evaluated on the same dataset. Moreover, also compared to the other systems, which have used other preprocessing methods, Clus-HMC-Ens is competitive: only the Funckenstein method and

GeneMANIA produce significantly better results on 3 and 2 evaluation measures, respectively. In a function-specific comparison over all 12 subsets (1846 functions in total), Clus-HMC-Ens still performed better than Funckenstein on 607 (according to AUPRC) and 625 (according to AUROC) functions, while it had an equal score for 98 (AUPRC) and 97 (AUROC) functions. Similarly, it performed better than GeneMANIA on 645/563 functions and had an equal score for 84/88 functions, respectively. This shows that none of the methods is guaranteed to be the best choice for any given function.

This comparison to the methods in the MouseFunc competition suggests that incorporating functional linkage information in the predictions made by an ensemble method can substantially improve its performance. Although we have taken a first step towards this by performing a trivial propositionalisation of protein-protein interaction data, a more direct approach could be designed. How this could be achieved for Clus-HMC-Ens is a nice direction for further work.

## 4.5 Conclusions

In this chapter, we have applied the HMC decision tree learner Clus-HMC to functional genomics data for *S. cerevisiae* and *A. thaliana*. We have also introduced Clus-HMC-Ens, the ensemble version of Clus-HMC and we have empirically shown that it outperforms state-of-the-art methods on *S. cerevisiae*, *A. thaliana* and *M. musculus* datasets.

First, we have shown that Clus-HMC outperforms an existing decision tree learner (C4.5H/M) w.r.t. predictive performance, while preserving the interpretability. Second, we have shown that the predictive performance boost in regular classification tasks obtained by using ensembles, carries over to the hierarchical multi-label classification context, in which the gene function prediction task is set. Third, by constructing an ensemble of Clus-HMC-trees, our method outperforms a statistical learner based on SVMs for *S. cerevisiae*, both in predictive performance and in efficiency. Fourth, this ensemble learner is competitive to statistical and network based methods for *M. musculus* data.

To summarise, Clus-HMC can give additional biological insight in the predictions. Moreover, Clus-HMC-Ens yields state-of-the-art quality for gene function prediction. The software implementing these methods is easy to use and available online as open-source software. As such, Clus-HMC(-Ens) are fast, highly automatic and have a high predictive performance. Therefore, Clus-HMC and Clus-HMC-Ens outperform the current state-of-the-art tools for gene function prediction.

Table 4.1: Comparison of AU($\overline{\text{PRC}}$) between Clus-HMC-Ens and the MouseFunc systems. For each of the 12 subsets, the AU($\overline{\text{PRC}}$) of Clus-HMC-Ens is compared with the MouseFunc systems. A win ($\oplus$) means that the MouseFunc system outperforms Clus-HMC-Ens, a loss ($\ominus$) means that it is outperformed by Clus-HMC-Ens.

| Subset | Clus-HMC-Ens | BSVM$^+$ | KLR | CSVM | GeneFAS | GeneMANIA | KIM | Funckenstein |
|---|---|---|---|---|---|---|---|---|
| BP_3-10 | 0.045 | 0.040$\ominus$ | 0.028$\ominus$ | 0.029$\ominus$ | 0.028$\ominus$ | 0.071$\oplus$ | 0.029$\ominus$ | 0.085$\oplus$ |
| BP_11-30 | 0.055 | 0.042$\ominus$ | 0.053$\ominus$ | 0.017$\ominus$ | 0.012$\ominus$ | 0.038$\ominus$ | 0.031$\ominus$ | 0.083$\oplus$ |
| BP_31-100 | 0.109 | 0.100$\ominus$ | 0.135$\oplus$ | 0.077$\ominus$ | 0.033$\ominus$ | 0.035$\ominus$ | 0.044$\ominus$ | 0.190$\oplus$ |
| BP_101-300 | 0.173 | 0.161$\ominus$ | 0.174$\oplus$ | 0.146$\ominus$ | 0.078$\ominus$ | 0.055$\ominus$ | 0.051$\ominus$ | 0.225$\oplus$ |
| CC_3-10 | 0.182 | 0.076$\ominus$ | 0.060$\ominus$ | 0.046$\ominus$ | 0.050$\ominus$ | 0.131$\ominus$ | 0.128$\ominus$ | 0.202$\oplus$ |
| CC_11-30 | 0.207 | 0.085$\ominus$ | 0.128$\ominus$ | 0.094$\ominus$ | 0.038$\ominus$ | 0.068$\ominus$ | 0.112$\ominus$ | 0.167$\ominus$ |
| CC_31-100 | 0.233 | 0.163$\ominus$ | 0.161$\ominus$ | 0.074$\ominus$ | 0.107$\ominus$ | 0.046$\ominus$ | 0.127$\ominus$ | 0.226$\ominus$ |
| CC_101-300 | 0.220 | 0.166$\ominus$ | 0.225$\oplus$ | 0.157$\ominus$ | 0.110$\ominus$ | 0.101$\ominus$ | 0.094$\ominus$ | 0.248$\oplus$ |
| MF_3-10 | 0.266 | 0.243$\ominus$ | 0.191$\ominus$ | 0.205$\ominus$ | 0.174$\ominus$ | 0.359$\oplus$ | 0.189$\ominus$ | 0.368$\oplus$ |
| MF_11-30 | 0.356 | 0.258$\ominus$ | 0.285$\ominus$ | 0.275$\ominus$ | 0.136$\ominus$ | 0.270$\ominus$ | 0.215$\ominus$ | 0.384$\oplus$ |
| MF_31-100 | 0.360 | 0.245$\ominus$ | 0.294$\ominus$ | 0.231$\ominus$ | 0.120$\ominus$ | 0.284$\ominus$ | 0.191$\ominus$ | 0.482$\oplus$ |
| MF_101-300 | 0.368 | 0.283$\ominus$ | 0.331$\ominus$ | 0.386$\oplus$ | 0.184$\ominus$ | 0.202$\ominus$ | 0.140$\ominus$ | 0.485$\oplus$ |

Table 4.2: Comparison of $\overline{\text{AUPRC}}$ between Clus-HMC-Ens and the MouseFunc systems. For each of the 12 subsets, the $\overline{\text{AUPRC}}$ of Clus-HMC-Ens is compared with the MouseFunc systems. A win ($\oplus$) means that the MouseFunc system outperforms Clus-HMC-Ens, a loss ($\ominus$) means that it is outperformed by Clus-HMC-Ens.

| Subset | Clus-HMC-Ens | BSVM$^+$ | KLR | CSVM | GeneFAS | GeneMANIA | KIM | Funckenstein |
|---|---|---|---|---|---|---|---|---|
| BP_3-10 | 0.120 | 0.156$\oplus$ | 0.075$\ominus$ | 0.075$\ominus$ | 0.108$\ominus$ | 0.170$\oplus$ | 0.108$\ominus$ | 0.198$\oplus$ |
| BP_11-30 | 0.110 | 0.141$\oplus$ | 0.087$\ominus$ | 0.085$\ominus$ | 0.074$\ominus$ | 0.151$\oplus$ | 0.107$\ominus$ | 0.162$\oplus$ |
| BP_31-100 | 0.139 | 0.172$\oplus$ | 0.158$\oplus$ | 0.140$\oplus$ | 0.094$\ominus$ | 0.177$\oplus$ | 0.116$\ominus$ | 0.244$\oplus$ |
| BP_101-300 | 0.171 | 0.172$\oplus$ | 0.169$\ominus$ | 0.173$\oplus$ | 0.104$\ominus$ | 0.160$\ominus$ | 0.056$\ominus$ | 0.214$\oplus$ |
| CC_3-10 | 0.319 | 0.249$\ominus$ | 0.119$\ominus$ | 0.083$\ominus$ | 0.233$\ominus$ | 0.324$\oplus$ | 0.271$\ominus$ | 0.316$\ominus$ |
| CC_11-30 | 0.260 | 0.194$\ominus$ | 0.212$\ominus$ | 0.151$\ominus$ | 0.131$\ominus$ | 0.235$\ominus$ | 0.178$\ominus$ | 0.267$\oplus$ |
| CC_31-100 | 0.217 | 0.232$\oplus$ | 0.197$\ominus$ | 0.161$\ominus$ | 0.191$\ominus$ | 0.261$\oplus$ | 0.144$\ominus$ | 0.287$\oplus$ |
| CC_101-300 | 0.244 | 0.217$\ominus$ | 0.259$\oplus$ | 0.221$\ominus$ | 0.177$\ominus$ | 0.258$\oplus$ | 0.118$\ominus$ | 0.279$\oplus$ |
| MF_3-10 | 0.320 | 0.441$\oplus$ | 0.258$\ominus$ | 0.228$\ominus$ | 0.427$\oplus$ | 0.465$\oplus$ | 0.304$\ominus$ | 0.472$\oplus$ |
| MF_11-30 | 0.356 | 0.373$\oplus$ | 0.347$\ominus$ | 0.393$\oplus$ | 0.350$\ominus$ | 0.401$\oplus$ | 0.302$\ominus$ | 0.455$\oplus$ |
| MF_31-100 | 0.269 | 0.289$\oplus$ | 0.230$\ominus$ | 0.278$\oplus$ | 0.242$\ominus$ | 0.291$\oplus$ | 0.255$\ominus$ | 0.416$\oplus$ |
| MF_101-300 | 0.322 | 0.317$\ominus$ | 0.321$\ominus$ | 0.374$\oplus$ | 0.295$\ominus$ | 0.391$\oplus$ | 0.172$\ominus$ | 0.441$\oplus$ |

Table 4.3: Comparison of $\overline{\text{AUROC}}$ between Clus-HMC-Ens and the MouseFunc systems. For each of the 12 subsets, the $\overline{\text{AUROC}}$ of Clus-HMC-Ens is compared with the MouseFunc systems. A win ($\oplus$) means that the MouseFunc system outperforms Clus-HMC-Ens, a loss ($\ominus$) means that it is outperformed by Clus-HMC-Ens.

| Subset | Clus-HMC-Ens | BSVM$^+$ | KLR | CSVM | GeneFAS | GeneMANIA | KIM | Funckenstein |
|---|---|---|---|---|---|---|---|---|
| BP_3-10 | 0.695 | 0.808$\oplus$ | 0.581$\ominus$ | 0.588$\ominus$ | 0.715$\oplus$ | 0.873$\oplus$ | 0.813$\oplus$ | 0.790$\oplus$ |
| BP_11-30 | 0.748 | 0.808$\oplus$ | 0.741$\ominus$ | 0.659$\ominus$ | 0.767$\oplus$ | 0.849$\oplus$ | 0.822$\oplus$ | 0.796$\oplus$ |
| BP_31-100 | 0.831 | 0.874$\oplus$ | 0.846$\oplus$ | 0.778$\ominus$ | 0.780$\ominus$ | 0.872$\oplus$ | 0.851$\oplus$ | 0.880$\oplus$ |
| BP_101-300 | 0.823 | 0.853$\oplus$ | 0.845$\oplus$ | 0.813$\ominus$ | 0.733$\ominus$ | 0.840$\oplus$ | 0.795$\ominus$ | 0.838$\oplus$ |
| CC_3-10 | 0.748 | 0.845$\oplus$ | 0.571$\ominus$ | 0.618$\ominus$ | 0.782$\oplus$ | 0.899$\oplus$ | 0.865$\oplus$ | 0.837$\oplus$ |
| CC_11-30 | 0.791 | 0.873$\oplus$ | 0.790$\ominus$ | 0.785$\ominus$ | 0.834$\oplus$ | 0.907$\oplus$ | 0.846$\oplus$ | 0.850$\oplus$ |
| CC_31-100 | 0.863 | 0.896$\oplus$ | 0.850$\ominus$ | 0.851$\ominus$ | 0.783$\ominus$ | 0.887$\oplus$ | 0.863 | 0.849$\ominus$ |
| CC_101-300 | 0.845 | 0.873$\oplus$ | 0.851$\oplus$ | 0.821$\ominus$ | 0.750$\ominus$ | 0.842$\ominus$ | 0.808$\ominus$ | 0.867$\oplus$ |
| MF_3-10 | 0.818 | 0.887$\oplus$ | 0.630$\ominus$ | 0.681$\ominus$ | 0.850$\oplus$ | 0.951$\oplus$ | 0.880$\oplus$ | 0.879$\oplus$ |
| MF_11-30 | 0.842 | 0.903$\oplus$ | 0.861$\oplus$ | 0.836$\ominus$ | 0.865$\oplus$ | 0.936$\oplus$ | 0.884$\oplus$ | 0.909$\oplus$ |
| MF_31-100 | 0.838 | 0.888$\oplus$ | 0.892$\oplus$ | 0.881$\oplus$ | 0.843$\oplus$ | 0.887$\oplus$ | 0.884$\oplus$ | 0.903$\oplus$ |
| MF_101-300 | 0.874 | 0.904$\oplus$ | 0.894$\oplus$ | 0.884$\oplus$ | 0.843$\ominus$ | 0.909$\oplus$ | 0.844$\ominus$ | 0.918$\oplus$ |

# Conclusion Part I

In this part, we have addressed the task of hierarchical multi-label classification (HMC) and its application to functional genomics.

In Chapter 3, we have compared three decision tree algorithms for HMC: (1) an algorithm that learns a single tree that predicts all classes at once (Clus-HMC), (2) an algorithm that learns a separate decision tree for each class (Clus-SC), and (3) an algorithm that learns and applies such single-label decision trees in a hierarchical way (Clus-HSC). We have evaluated the algorithms on 24 datasets from functional genomics.

As expected, Clus-HMC outperforms Clus-SC and Clus-HSC in terms of efficiency and model size. Indeed, learning a single HMC tree takes more time than learning a single-label tree, but it is much faster than learning $|C|$ single-label trees (with $|C|$ the number of classes in the hierarchy), even when these are learned hierarchically. Equivalently, a single HMC tree might be larger than a single SC or HSC tree, but it is much smaller than the union of the $|C|$ single-label trees. Moroever, interpreting one HMC tree is also easier than interpreting $|C|$ single-label trees, while an HMC tree also identifies global features with a high overall relevance.

More surprisingly, the HMC approach is superior in terms of predictive performance. Intuitively, one expects Clus-SC and Clus-HSC to perform better, since a separate model is learned for each class. Experimental analysis has shown, however, that these approaches suffer from overfitting, while Clus-HMC is less prone to it. In hindsight, this makes sense, since is it much harder for a model to overfit for $|C|$ classes at once, rather than for just one.

In Chapter 4, we have presented Clus-HMC-Ens, which is the ensemble version of Clus-HMC, and compared both Clus-HMC and Clus-HMC-Ens to state-of-the-art methods for functional genomics found in the biomedical literature. We have evaluated the different algorithms on three model organisms from biology.

First, we have shown that the predictive performance boost in regular classification tasks obtained by using ensembles, carries over to the HMC context. There is a clear trade-off to be considered, however, between the increase in pre-

dictive performance on the one hand and the time to learn the model and its interpretability on the other hand.

Second, we have shown that CLUS-HMC outperforms C4.5H/M, an existing decision tree learner for HMC, w.r.t. predictive performance, while preserving the interpretability. An explanation for this could be that, instead of just giving predicted functions as output, CLUS-HMC outputs a probability score for each gene-function pair, to which a suitable classification threshold can be applied afterwards, and the fact that CLUS-HMC preserves the hierarchy constraint, making sure that the parents classes of a predicted class are always predicted as well.

Third, CLUS-HMC-ENS outperforms statistical learning methods such as SVMs w.r.t. efficiency, interpretability and ease of use, while having a competitive predictive performance.

To summarise, CLUS-HMC and CLUS-HMC-ENS are state-of-the-art tools for gene function prediction and therefore, we believe they should be considered for making automated predictions in functional genomics.

# Part II

# Structured Input Learning for the Prediction of Molecular Activity

# Outline Part II

In the second part of the thesis, we focus on graph mining, which is concerned with learning techniques that have graphs as their input. The application on which we focus is the learning of structure-activity relationships (SAR). The goal of SAR is to predict the activity of molecules based on their atom-bond structure. The motivation for this is that molecules with a similar structure tend to have the same function [Johnson and Maggiora, 1990].

An important step towards the discovery of new drugs is the identification of molecules that play an active role in the regulation of biological processes or disease states. Even though it has become easier during the last years to acquire molecular data, this process, wich is often referred to as screening, is still costly and time consuming. For this reason, pharmaceutical companies have become interested in SAR techniques, that can predict the activity of molecules automatically and can therefore seriously decrease the amount of molecules that should be screened in the lab. Various properties of molecules are of interest in drug development, such as toxicity, carcinogenicity or biodegradability. An ideal drug molecule for a disease is the one that is most active against the disease, while at the same time it does not have undesirable properties such as toxicity or carcinogenicity, which could cause unwanted side effects.

Because graphs provide a natural representation of the atom-bond structure of molecules, they have become very popular in the context of SAR: each vertex of the graph can represent an atom, while an edge represents a molecular bond [Raymond and Willett, 2002b]. However, algorithms on graphs that aim at using all available structural information often involve the matching of subgraphs or other combinatorial operations trying to align graphs optimally.

In this part, we will propose new graph mining algorithms that can be used for SAR learning tasks. In Chapter 5, we will introduce an algorithm that computes a maximum common subgraph (MCS) between a pair of graphs efficiently thanks to the restriction to outerplanar graphs and the block-and-bridge-preserving subgraph isomorphism. This algorithm will be used in two learning techniques for SAR.

In Chapter 6, we will show that a metric can be constructed from the MCS

algorithm. Metrics are important for structural similarity search among small molecules, which is the standard tool used for virtual screening and in silico drug development. The task then comes down to finding an appropriate similarity measure between molecules. Such a structural similarity measure should ideally fulfil two requirements: (1) it should be efficiently computable, which is important when analysing large molecular databases, and (2) the notion of similarity should discriminate between molecules w.r.t. the activity of interest. Finding such similarity measures is one of the current challenges in chemoinformatics [Ceroni et al., 2007; Deshpande et al., 2005]. But (relational) metrics for graphs are also interesting for other application domains.

In Chapter 7, we will use the MCS algorithm to generate features for graphs. These features can then be used for the classification of molecules [Deshpande et al., 2005; Wale et al., 2008]. It is important that the features can be generated efficiently and that they preserve the molecular structure as much as possible, while at the same time having a good predictiveness.

# Chapter 5

# A Polynomial-time Maximum Common Subgraph Algorithm

## 5.1 Introduction

In this chapter, we will present an algorithm that computes a maximum common subgraph (MCS) of a pair of outerplanar graphs. Intuitively, an MCS of two graphs can be described as a subgraph common to both graphs, such that no other common subgraph has a size that is strictly larger. Note that, in theory, there can exist an exponential number of them. An example of an MCS of two molecular graphs is shown in Fig. 5.1.

The reason that we are interested in them is threefold. First, an MCS of two molecular graphs may contain shared properties that relate to their molecular activity. Second, unlike most descriptor-based methods for molecules, MCSs are based on structural information and are able to reflect local similarities in graphs rather than global ones. Third, because of the visual nature of graphs, chemists can visually investigate the MCSs that are predicted to be responsible for certain activities. Moreover, from previous work it is known that methods based on computing MCSs work well on several structure-activity learning tasks [Raymond and Willett, 2002b].

Finding the MCSs of two graphs, however, is known to be an NP-hard problem [Garey and Johnson, 1979]. Nevertheless, previous theoretical work on graphs has shown that in many cases, when certain constraints on the structure of the graphs hold, efficient matching algorithms exist [Mitchell, 1979; Lingas, 1989]. Sequences, trees and outerplanar graphs (the latter are graphs which can be embedded in the

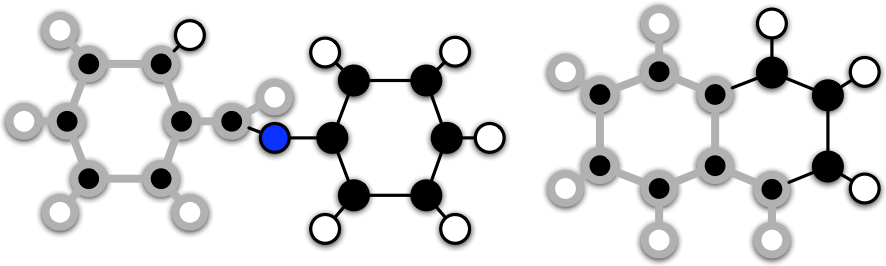

Figure 5.1: A maximum common subgraph (MCS) of two molecular graphs (highlighted in gray).

plane in such a way that all of their vertices lie on the boundary of the outer face) are examples of such subclasses of general graphs. Investigation on the NCI[1] database, a collection of datasets containing over 250,000 chemical compounds, revealed that only 8.8% of these graphs are trees, while 94.5% of them are outerplanar [Horváth et al., 2006]. Hence, algorithms that can deal with outerplanar graphs will be able to efficiently handle the majority of the molecules, while at the same time they will be much more practical than those handling trees.

Recently, a new "block-and-bridge-preserving" (BBP) matching operator for outerplanar graphs was introduced [Horváth et al., 2006]. It is based on the idea that it only makes sense to match "similar" parts of the graphs. It was shown that, using this operator, the computation of pattern embeddings, that is, checking whether a pattern is subgraph isomorphic to graphs in the dataset, is polynomial, such that frequent patterns can be mined efficiently in outerplanar graphs. Using this idea of the BBP subgraph isomorphism, we propose a polynomial MCS algorithm for outerplanar graphs. Here, the task is to find a maximal graph that is subgraph isomorphic to two given graphs. The MCS algorithm can be used as a component in several classification algorithms, which will be presented in the next chapters.

The most important contributions of this chapter are the following:

- We provide a detailed technical description of the algorithm that computes an MCS under the BBP subgraph isomorphism, with a proof of its correctness and an analysis of its time complexity.

- We compare the efficiency of our algorithm to a state-of-the-art algorithm that computes an MCS under the general subgraph isomorphism.

[1] National Cancer Institute (`http://cactus.nci.nih.gov/`).

The contents of this chapter are the result of joint work with Jan Ramon and Maurice Bruynooghe, partially published in [Schietgat et al., 2007].

The chapter is organised as follows. We start by presenting some necessary graph theoretical concepts in Sect. 5.2. Section 5.3 presents the MCS algorithm, while Sect. 5.4 discusses related work. In Sect. 5.5, we present an empirical study of the algorithm's complexity and in Sect. 5.6, we draw conclusions.

## 5.2 Graph theoretical concepts

This section gives the relevant definitions necessary to understand the algorithm that computes an MCS of two outerplanar graphs. For an overview of graph theory, we refer to Diestel [2000]. We start by repeating the definition of a graph that was introduced in Sect. 2.2.3.

**Definition 5.1** *A labeled graph is a quadruple $G(V, E, \Sigma, \lambda)$, with $V$ a finite set of vertices and $E \subseteq \{\{u, v\} \mid u, v \in V\}$ a set of edges. $\Sigma$ is a finite set of labels and $\lambda : V \cup E \rightarrow \Sigma$ is a function assigning a label to each element of $V \cup E$.*

We will denote the set of all graphs with $\mathcal{G}$. In this text, we only consider simple graphs (for which between every two vertices there is at most one edge) with undirected edges. Therefore, we will use the set notation $\{u, v\}$, which does not imply an ordering between $u$ and $v$.

**Definition 5.2** *The size $|\cdot| : \mathcal{G} \rightarrow \mathbb{R}$ of a graph is a function mapping a graph to a real number of the form $|G| = \sum_{x \in V(G) \cup E(G)} w_{\lambda_G(x)}$, where each possible label of $l \in \Sigma$ has been assigned a positive weight $w_l$.*

**Example 5.1** *Assume that all vertices of $v \in V(G)$ have the same label $\lambda_G(v) = l_1$ and all the edges $e \in E(G)$ have the same label $\lambda_G(e) = l_2$. Let $w_{l_1} = 1$ and $w_{l_2} = 0$. Then, the corresponding size function maps every graph $G$ on the number of vertices of $G$.*

**Definition 5.3** *The open neighborhood of a vertex $v$ in a graph $G$, denoted $N_G^o(v)$, is the set of all the vertices adjacent to $v$, i.e. $N_G^o(v) = \{x \mid \{v, x\} \in E(G)\}$.*

**Definition 5.4** *A sequence $x_0, x_1, \ldots, x_n$ of vertices is a path from $x_0$ to $x_n$ iff $\{x_i, x_{i+1}\} \in E(G)$, for all $i \in [0, n-1]$. A cycle $x_0, \ldots, x_n$ is a path such that $x_0 = x_n$. A path without repeated vertices is a simple path; a cycle without repeated vertices apart from the start and end vertex is a simple cycle.*

**Definition 5.5** *A graph $G$ is connected if there is a path between any pair of its vertices; it is biconnected if for any two vertices $u$ and $v$ of $G$, there is a simple cycle containing $u$ and $v$.*

**Definition 5.6** *A graph is planar if it has a planar embedding, i.e. it can be drawn in the plane in such a way that no two edges intersect except at a vertex in common. The regions formed by the edges in a planar embedding are called faces. There is one unbounded face, which is called the outer face.*

**Definition 5.7** *A tree $T$ is a graph for which there is a unique path between every pair of vertices $u, v \in T$.*

**Definition 5.8** *Let $G$ and $H$ be graphs. $G$ is a subgraph of $H$, if (i) $V(G) \subseteq V(H)$, (ii) $E(G) \subseteq E(H)$, and (iii) $\lambda_G(x) = \lambda_H(x)$ holds for every $x \in V(G) \cup E(G)$.*

**Definition 5.9** *A biconnected component or block of a graph $G$ is a maximal subgraph of $G$ that is biconnected.*

**Definition 5.10** *In a graph $G$, a bridge is an edge of $G$ that does not belong to a block.*

We denote with $Bl_i(G)$ all edges of the graph $G$ that belong to block $B_i$, with $Bl(G)$ all the edges that are involved in blocks of $G$ and with $Br(G)$ all the bridges of $G$.

**Definition 5.11** *An outerplanar graph is a planar graph which can be embedded in the plane in such a way that all of its vertices lie on the boundary of the outer face.*

We denote the set of all outerplanar graphs with $\mathcal{G}_{op}$. An outerplanar graph consists entirely of blocks and bridges, that is, its edges can be partitioned as: $E(G) = Br(G) \cup Bl_1(G) \cup \ldots \cup Bl_i(G)$ with $Br(G) \cap Bl_1(G) \cap \ldots \cap Bl_i(G) = \emptyset$. Moreover, bridges are always adjacent to the outer face, while block edges are adjacent to two faces, one of which can possibly be the outer face. Two blocks can share a vertex, but no edge. In this case, the blocks are not maximal as their union would be larger and still biconnected.

**Example 5.2** *Fig. 5.2a shows an example of an non-outerplanar graph. Indeed, there is one vertex (labeled $v$) that is not on the outside of the graph. The graphs if Fig. 5.2b and Fig. 5.2c are, however, outerplanar. Every vertex is labeled with a color representing a chemical element: black for carbon, white for hydrogen and blue for nitrogen. Note that in all graphs, the edges between carbons and hydrogens are bridges, while the rest of the edges are involved in blocks. From a chemical viewpoint, blocks correspond to ring structures while bridges are linear fragments of the molecule.*

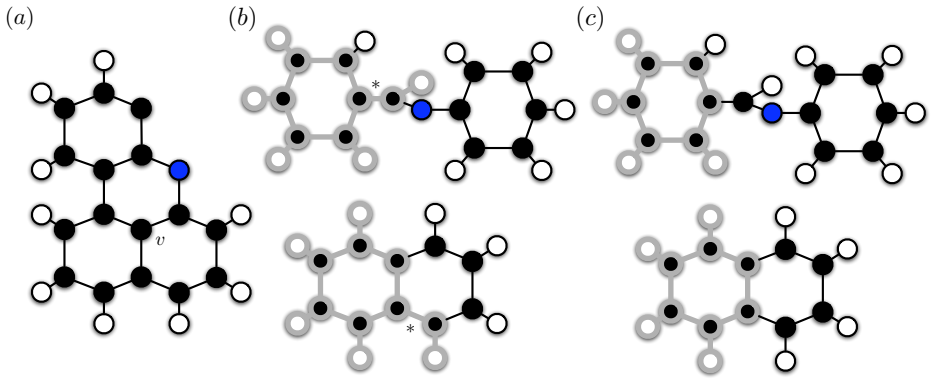


Figure 5.2: Examples of molecular graphs. The colors of the vertices correspond to their labels: black for carbon, white for hydrogen and blue for nitrogen. (a) Example of a non-outerplanar graph, with the vertex that is not on the outside of the graph labeled $v$. (b) Two molecules with their maximum common subgraph computed under the general subgraph isomorphism ($\text{MCS}_{\preceq}$), highlighted in gray. (c) The same two molecules with their maximum common subgraph computed under the BBP subgraph isomorphism ($\text{MCS}_{\sqsubseteq}$), highlighted in gray.

**Definition 5.12** *Two graphs $G$ and $H$ are isomorphic under $\varphi$ if there exists a bijection $\varphi : V(G) \to V(H)$ such that for every $u, v \in V(G)$ the following holds: (i) $\{u, v\} \in E(G)$ iff $\{\varphi(u), \varphi(v)\} \in E(H)$, (ii) $\lambda_G(u) = \lambda_H(\varphi(u))$, and (iii) if $\{u, v\} \in E(G)$ then $\lambda_G(\{u, v\}) = \lambda_H(\{\varphi(u), \varphi(v)\})$.*

Isomorphism is an equivalence relation on $\mathcal{G}$. We will denote the set of all equivalence classes under isomorphism with $\mathcal{G}^{\equiv}$.

**Definition 5.13** *A graph $G$ is subgraph isomorphic to $H$ under $\varphi$, denoted $G \preceq_{\varphi} H$, iff $G$ is isomorphic to a subgraph of $H$ under $\varphi$.*

If the bijection $\varphi$ is clear from the context, we omit it from the notation, i.e. $G \preceq H$. The subgraph isomorphism problem, which decides whether $G$ is subgraph isomorphic to $H$ is known to be NP-complete [Garey and Johnson, 1979]; this also holds for outerplanar graphs [Syslo, 1982].

**Definition 5.14** *A block-and-bridge-preserving (BBP) subgraph isomorphism under $\varphi$ from $G$ to $H$ is a subgraph isomorphism under $\varphi$ from $G$ to $H$, such that $\forall\{u, v\} \in E(G)$: if $\{u, v\} \in Br(G)$, then $\{\varphi(u), \varphi(v)\} \in Br(H)$ and if $\{u, v\} \in Bl(G)$, then $\{\varphi(u), \varphi(v)\} \in Bl(H)$.*

We denote that a graph $G$ is BBP subgraph isomorphic to $H$ by $G \sqsubseteq_{\varphi} H$. Again, we omit the subscript $\varphi$ if not needed. The BBP subgraph isomorphism is a special case of the general subgraph isomorphism, requiring an additional constraint stating that bridges of $G$ are only mapped to bridges of $H$ and block edges of $G$ only to block edges of $H$. Contrary to the subgraph isomorphism problem, the BBP subgraph isomorphism problem is computable in polynomial time for outerplanar graphs [Horváth et al., 2006]. For trees, which form a subset of outerplanar graphs (i.e., they are block-free), the BBP subgraph isomorphism is equivalent to the subtree isomorphism.

**Definition 5.15** *A common subgraph $I$ of two graphs $G$ and $H$ is a connected graph such that $I \preceq G$ and $I \preceq H$; it is a maximum common subgraph (MCS) when in addition there exists no other common subgraph $J$, such that $size(I) < size(J)$.*

We use the notation $\text{MCS}_{\preceq}$ to indicate that the MCS is computed under the general subgraph isomorphism and implicitly assume that it is always connected. In the same way, we define the $\text{MCS}_{\sqsubseteq}$. Note that, since the BBP subgraph isomorphism is a restricted version of the general subgraph isomorphism, an $\text{MCS}_{\sqsubseteq}$ is subgraph isomorphic to one of the $\text{MCSs}_{\preceq}$. Note also that, in the worst case, there may exist a potentially exponential number of MCSs. Finally, we remark that, when computing an $\text{MCS}_{\sqsubseteq}$, it is not necessary that a complete block belonging to one of the graphs is part of an MCS, since a block can consist of multiple biconnected components.

Computing the $\text{MCS}_{\preceq}$ or the $\text{MCS}_{\sqsubseteq}$ of two general graphs is NP-hard [Garey and Johnson, 1979]. In this chapter, however, we will show that it is possible to compute one $\text{MCS}_{\sqsubseteq}$ of two outerplanar graphs in polynomial time by using the block-and-bridge-preserving (BBP) subgraph isomorphism [Horváth et al., 2006]. From an application point of view, the BBP subgraph isomorphism can be motivated from the fact that in molecules cyclic structures and linear fragments usually behave differently, and hence treating them separately might well be a good thing.

Figure 5.2 shows a comparison between the $\text{MCS}_{\preceq}$ (b) and the $\text{MCS}_{\sqsubseteq}$ (c). In both examples, the MCS is highlighted in gray. Note that one of the edges is a bridge in the upper graph, while it belongs to a block in the lower graph (marked with a * in both graphs) and hence, it cannot be mapped under the BBP subgraph isomorphism. Chemists consider it relevant not to map linear fragments with fragments that are part of a ring structure.

We also give some additional definitions w.r.t. matchings, as described in [Munkres, 1957].

**Definition 5.16** *A matching $f$ between sets $A$ and $B$ is a relation such that for all $(a_1, b_1)$, $(a_2, b_2) \in f$: $a_1 = a_2$ iff $b_1 = b_2$, so each element of $A$ is associated with at most one element of $B$ and vice versa.*

**Definition 5.17** *The weighted maximal matching problem, also known as the assignment problem, is an optimisation problem where two sets $A$ and $B$ are given together with a weight function $w : A \times B \to \mathbb{R}$ and the task is to find a matching $m$ between $A$ and $B$ such that $\sum_{(a,b)\in m} w(a,b)$ is maximal.*

Computing a weighted maximal matching can happen in $O(V^2E)$, with $V = |A| + |B|$ and $E$ the number of edges between $A$ and $B$.

From now on, we will simply use MCS to indicate the $\text{MCS}_{\sqsubseteq}$.

## 5.3 Computing a maximum common subgraph of two outerplanar graphs

Our algorithm to compute an MCS of two outerplanar graphs $G$ and $H$ is based on a dynamic programming strategy and consists of two steps. First, we will generate specific subgraphs (which we call *relevant subgraphs*) of $G$ and $H$ and define their parent-child relationships. Second, we will compute an MCS for each pair of generated relevant subgraphs in a bottom-up way, building on the already computed solutions for pairs of their children. In this way, we obtain a solution for an MCS of the original graphs $G$ and $H$.

In Sect. 5.3.1, we will introduce some definitions and notations. In Sect. 5.3.2, we show how to generate the relevant subgraphs from an outerplanar graph and how to determine the parent-child relationships between the relevant subgraphs. Finally, in Sect. 5.3.3, we show how to efficiently compute an MCS of pairs of relevant subgraphs in a bottom-up fashion.

### 5.3.1 Definitions and notations

Given an outerplanar graph $G$, we define the two kinds of relevant subgraphs of $G$: the *block-preserving-subgraphs* and the *block-splitting-subgraphs*. Intuitively, the former are subgraphs in which a block is either entirely included in the subgraph or not; the latter are subgraphs which are created by removing part of a block between two vertices. The reason why we need those particular kinds of subgraphs will become clear in the second step of the algorithm (Sect. 5.3.3), in which solutions for relevant subgraphs are computed.

#### 5.3.1.1 Block-preserving-subgraphs

**Definition 5.18 (BPS)** *A subgraph $S$ of an outerplanar graph $G$ is a block-preserving-subgraph (BPS) of $G$ if and only if either $S = G$ or there exists a vertex $r \in V(G)$ and an edge $e \in E(G)$ with $r \in e$ such that $S$ is the maximal connected graph of $G$ containing $r$ for which the following holds:*

- *$S$ does not contain $e$,*

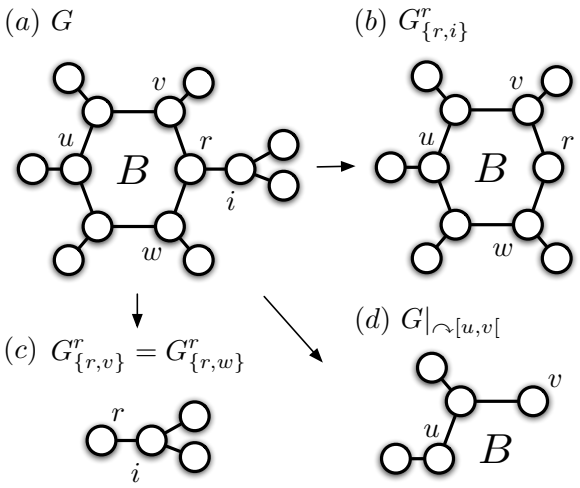


Figure 5.3: (a) An outerplanar graph $G$. (b) Its BPS $G^r_n$. (c) Its BPS $G^r_B$. (d) Its BSS $G|_{\curvearrowright[u,v[}$.

- $S \sqsubseteq G$.

We call $r$ the root of the BPS and denote $S$ as $G^r_e$. The choice of root is important because this will determine what the children relevant subgraphs of the BPS will be. When the edge $e$ of a BPS $G^r_e$ is not important, we simply denote the BPS as $G^r$. When $S = G$, we denote it as $G^r_{\{r,r\}}$ with $r \in V(G)$. The requirement that $S \sqsubseteq G$ implies that, if $e \in Bl_i(G)$, then none of the edges of $Bl_i(G)$ will belong to $S$. Otherwise, the block would not be preserved.

Note that this definition allows for the construction of the same BPSs. For example, when $e_1$ and $e_2$ are part of the same block, that is, $e_1, e_2 \in Bl_i(G)$, then $G^r_{e_1}$ and $G^r_{e_2}$ are the same BPSs. In order to solve this issue, it suffices to check explicitly whether two edges $e_1$ and $e_2$ belong to the same block. In this context, we also introduce an additional notation $G^r_{Bl_i(G)}$ for $G^r_e$, which does not specify the edge $e$ and only indicates that $e$ is part of the block $Bl_i(G)$.

Figures 5.3, 5.4 and 5.5 show examples of BPSs of several outerplanar graphs $G$. We denote with $BPS(G)$ the set of all BPSs of an outerplanar graph $G$.

#### 5.3.1.2 Block-splitting-subgraphs

Every block of an outerplanar graph $G$ has a unique Hamiltonian cycle over its vertices [Mitchell, 1979]. Given a planar embedding of $G$, we can either number the block's vertices according to the clockwise ($\curvearrowright$) or the counterclockwise ($\curvearrowleft$)

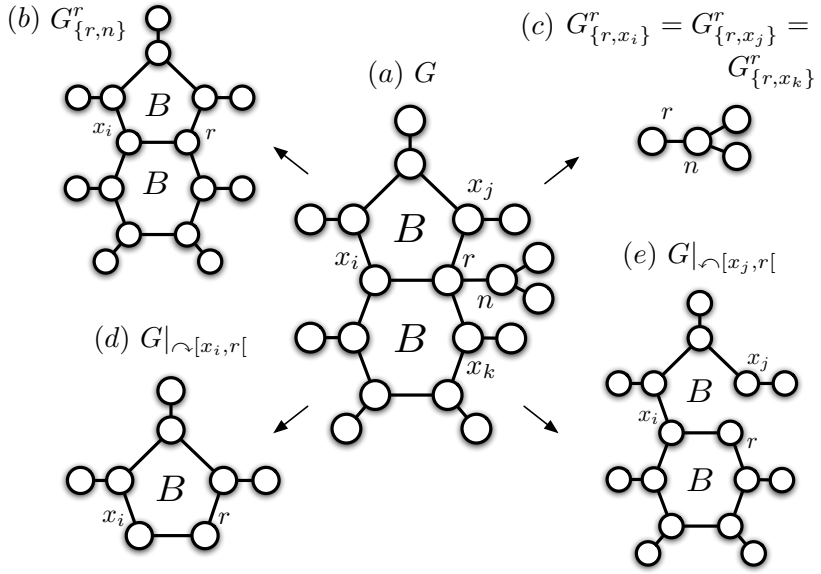


Figure 5.4: (a) An outerplanar graph $G$. (b) Its BPS $G^r_n$. (c) Its BPS $G^r_{x_i}$. (d) Its BSS $G|_{\curvearrowright[x_i,r[}$. (d) Its BSS $G|_{\curvearrowleft[x_j,r[}$.

orientation. In the context of a block $B$, if $o \in \{\curvearrowright, \curvearrowleft\}$ is an orientation and $u \in V(B)$ is a vertex, we denote with $u + o(i)$ the vertex going $i$ steps in the orientation $o$ along the Hamiltonian path. If $u, v \in V(B)$ are distinct vertices, we will denote with $o[u, v[$ the sequence of vertices in the orientation $o$ along the Hamiltonian path of $B$ between (and including) $u$ and $v$. Notice that $o[u, v[$ is uniquely defined by the orientation $o$, the begin point $u$ and the end point $v$, as a pair of vertices can only have at most one common block. We call such a sequence $o[u, v[$ a block interval. We say that $o[x, y[ \subseteq o[u, v[$ iff $x, y \in o[u, v[$ and $x$ comes before $y$ in $o[u, v[$ according to the orientation $o$.

**Definition 5.19 (BSS)** *A subgraph $S$ of an outerplanar graph $G$ is a block-splitting-subgraph (BSS) of $G$ if and only if there exists a pair of vertices $u$ and $v$ belonging to a block $B$ of $G$ ($u \neq v$) and an orientation $o$ such that $S$ is the maximal connected subgraph containing $u$ and $v$ for which the following holds:*

- *$S$ does not contain vertices of $V(B) \setminus o[u, v[$,*
- *$S$ does not contain bridges adjacent to $v$,*
- *$S$ does not contain edges of blocks different from $B$ adjacent to $v$.*

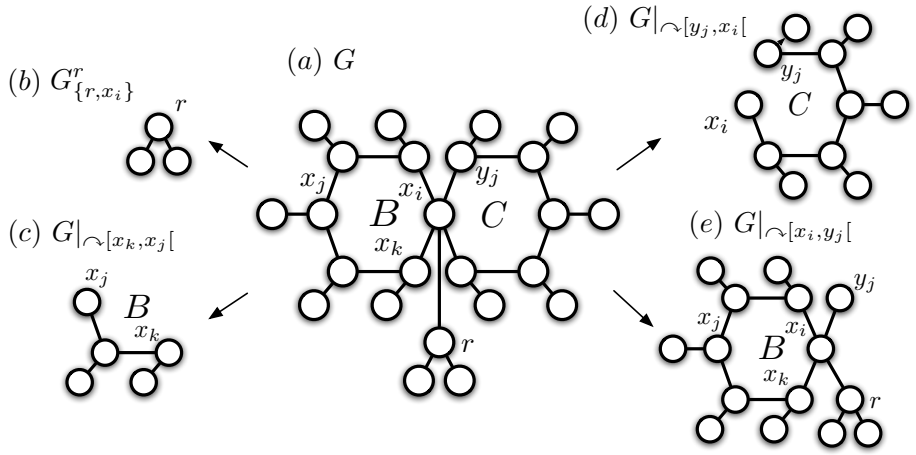


Figure 5.5: (a) An outerplanar graph $G$. (b) Its BPS $G^r_{x_i}$. (c) Its BSS $G|_{\curvearrowright[x_k,x_j[}$. (d) Its BSS $G|_{\curvearrowright[y_j,x_i[}$. (d) Its BSS $G|_{\curvearrowright[x_i,y_j[}$.

We denote $S$ as $G|_{o[u,v[}$ and call $v$ the root of the BSS. We use the notation "$o[u,v[$" to stress that the bridges or edges of other blocks adjacent to $u$ and all vertices between $u$ and $v$ are kept, while the bridges or edges of other blocks adjacent to $v$ are removed. Note that $G|_{o[u,u[} = G$ and that $G = G|_{o[u,v[} \cup G|_{o[v,u[}$. Note also that if an outerplanar graph does not have blocks, there are no BSSs.

Figures 5.3, 5.4 and 5.5 show examples of BSSs of outerplanar graphs $G$. We denote with $BSS(G)$ the set of all BSSs of an outerplanar graph $G$.

We also define the specific MCSs of pairs of relevant subgraphs. On the one hand, we define a BPS-MCS between two BPSs in which the root vertices are mapped and on the other hand, we define a BSS-MCS between two BSSs in which the begin and end vertices are mapped. As will become clear in the next section, this is sufficient to ensure that the solution of the algorithm contains a valid MCS.

**Definition 5.20 (BPS-MCS)** *If $G$ and $H$ are two outerplanar graphs embedded in the plane, and $G^r$ and $H^s$ are two of their respective BPSs, then*

$$\sigma(G^r, H^s) = \max\{S^t \mid S^t \sqsubseteq_{\varphi_G} G^r \wedge\ S^t \sqsubseteq_{\varphi_H} H^s\}$$

*with BBP subgraph isomorphism mappings $\varphi_G : S^t \to G^r$ and $\varphi_H : S^t \to H^s$ such that $\varphi_G(t) = r$ and $\varphi_H(t) = s$ and the function max selecting one of these graphs with maximum size.*

In words, $\sigma$ contains an MCS of the two BPSs $G^r_{\{r,i\}}$ and $H^s_{\{s,j\}}$ given that their root vertices are mapped onto each other. It is important to note that in the case there are multiple solutions, only one will be selected randomly.

**Definition 5.21 (BSS-MCS)** *If $G$ and $H$ are two outerplanar graphs embedded in the plane, and $G|_{o[u,v[}$ and $H|_{o'[u',v'[}$ are two of their respective BSSs, then*

$$\sigma(G|_{o^G[r',r[}, H|_{o^H[s',s[}) = \max\{S|_{o^S[t',t[} \mid S|_{o^S[t',t[} \sqsubseteq_{\varphi_G} G|_{o^G[r',r[} \wedge$$

$$S|_{o^S[t',t[} \sqsubseteq_{\varphi_H} H|_{o^H[s',s[}\}$$

*with BBP subgraph isomorphism mappings $\varphi_G : S|_{o^S[t',t[} \to G|_{o^G[r',r[}$ and $\varphi_H : S|_{o^S[t',t[} \to H|_{o^H[s',s[}$ such that $\varphi_G(t') = r'$, $\varphi_G(t) = r$, $\varphi_H(t') = s'$ and $\varphi_H(t) = s$.*

In words, $\sigma$ contains an MCS of $G|_{o^G[r',r[}$ and $H|_{o^H[s',s[}$, given the correspondence of their begin and end vertices. Also here only one solution will be selected in the case there are multiple possibilities.

### 5.3.2 Generation of relevant subgraphs and computation of the parent-child relationships

First, given two outerplanar graphs $G$ and $H$, we will specify which relevant subgraphs will be generated. We call a vertex a leaf if it either is adjacent to a single bridge or belongs to a block. For $G$ we will generate the BPSs $G^r_e$ for every $r \in V(G)$ that is a leaf and every $e \in E(G) \cup \{r, r\}$ adjacent to $r$. For $H$ it suffices to generate the BPSs $H^s_f$ for a random $s \in V(H)$ that is a leaf and every $f \in E(H) \cup \{s, s\}$ adjacent to $s$.

Similarly, for $G$ we will generate the BSSs $G|_{o[u,v[}$ for every $u, v$ that are vertices of a same block of $G$ and both orientations ($o \in \{\curvearrowleft, \curvearrowright\}$), while for $H$ it suffices to generate BSSs $H|_{o[u',v'[}$ for a single orientation.

Next, we show how to compute the parent-child relationships between the relevant subgraphs. The reason why this is necessary is that the dynamic programming strategy of the second step requires one to have access to children of relevant subgraphs in order to compute incremental solutions.

We define a parent-child relationship between the relevant subgraphs by exploiting the notation introduced for them.

**Definition 5.22 (Children of a BPS $G^r$)** *There are three cases:*

1. *If $\{r, i\}$ is a bridge of $G^r$, then $G^i_{\{r,i\}}$ is a BPS child of $G^r$, that is, the subgraph with root $i$ that remains after removing the bridge. Note that $i$ is the root of the child.*

2. *If $B$ is a block in $G^r$ that contains $r$ as one of its vertices, then $G^r_B$ is a BPS child of $G^r$, that is, the subgraph with root $r$ that remains after removing all vertices of $B$ but $r$. Note that $r$ is the root of the child.*

3. *If $B$ is a block in $G^r$ that (a) contains $r$ as one of its vertices, (b) $o$ is an orientation and $\{r, x_0, \ldots, x_n, r\}$ is the Hamiltonian cycle according to $o$, and (c) has $\{r, x_i\}$ (with $i < n$) as one of its edges, then $G|_{o[x_i,r[}$ is a BSS child of $G^r$. Note that $r$ is the root of the child.*

**Definition 5.23 (Children of a BSS $G|_{o[x_i,r[}$)** *We assume that $G|_{o[x_i,r[}$ is splitting a block $B$ with Hamiltonian cycle $\{r, x_0, \ldots, x_n, r\}$. There are two cases:*

1. *$G_B^{x_i}$ is a BPS child of $G|_{o[x_i,r[}$. Note that $x_i$ is the root of the child.*
2. *If $\{x_i, x_k\}$ is an edge of the block $B$, part of $G|_{o[x_i,r[}$, then $G|_{o[x_k,r[}$ is a BSS child of $G|_{o[x_i,r[}$. Note that $r$ is the root of the child.*

Figure 5.6 shows examples of BPS and BSS children.

We also define two structures that contain the relevant subgraphs and the parent-child relationships between them.

**Definition 5.24 (Relevant subgraph DAG)** *The relevant subgraph DAG of an outerplanar graph $G$ has as root $G$ itself. Its only child is $G^r$ with $r$ a random leaf of $G$. For all vertices of the DAG except the root, the children are the relevant subgraphs defined according to Definitions 5.22 and 5.23, except for the BSS children, where only one orientation is randomly selected.*

**Definition 5.25 (Extended relevant subgraph DAG)** *The extended relevant subgraph DAG of an outerplanar graph $G$ has as root $G$ itself. It has a child $G^r$ for every $r$ that is a leaf of $G$. For all vertices of the DAG except the root, the children are the relevant subgraphs defined according to Definitions 5.22 and 5.23. Note that both orientations are used to create BSS children.*

Both the relevant subgraph DAG as the extended relevant subgraph DAG can contain doubles. However, because a BPS $G_e^r$ is uniquely defined by $r$ and $e$ and a BSS $G|_{o[u,v[}$ is uniquely defined by $o$, $u$, and $v$, these doubles can easily be recognised.

**Proposition 5.1** *Let $S$ be a BPS of $G$ defined by the mapping $\varphi : S \to G$. The relevant subgraph DAG of $G$ contains a vertex with a relevant subgraph $R^s$ such that $s$ is in the range of $\varphi$ and $\varphi(S)$ is a subgraph of $R^s$.*

Proof:

- Let $G^r$ be the child of the root of the DAG. Observe that $\varphi(S)$ is a subgraph of $G^r$. If $r$ is in the range of $\varphi$, then the condition is trivially satisfied.
- Otherwise, also the edges adjacent to $r$ are not part of $\varphi(S)$. Hence, because $S$ is connected, $\varphi(S)$ must be a subgraph of one of the children. Let $G'^{r'}$ be this child.

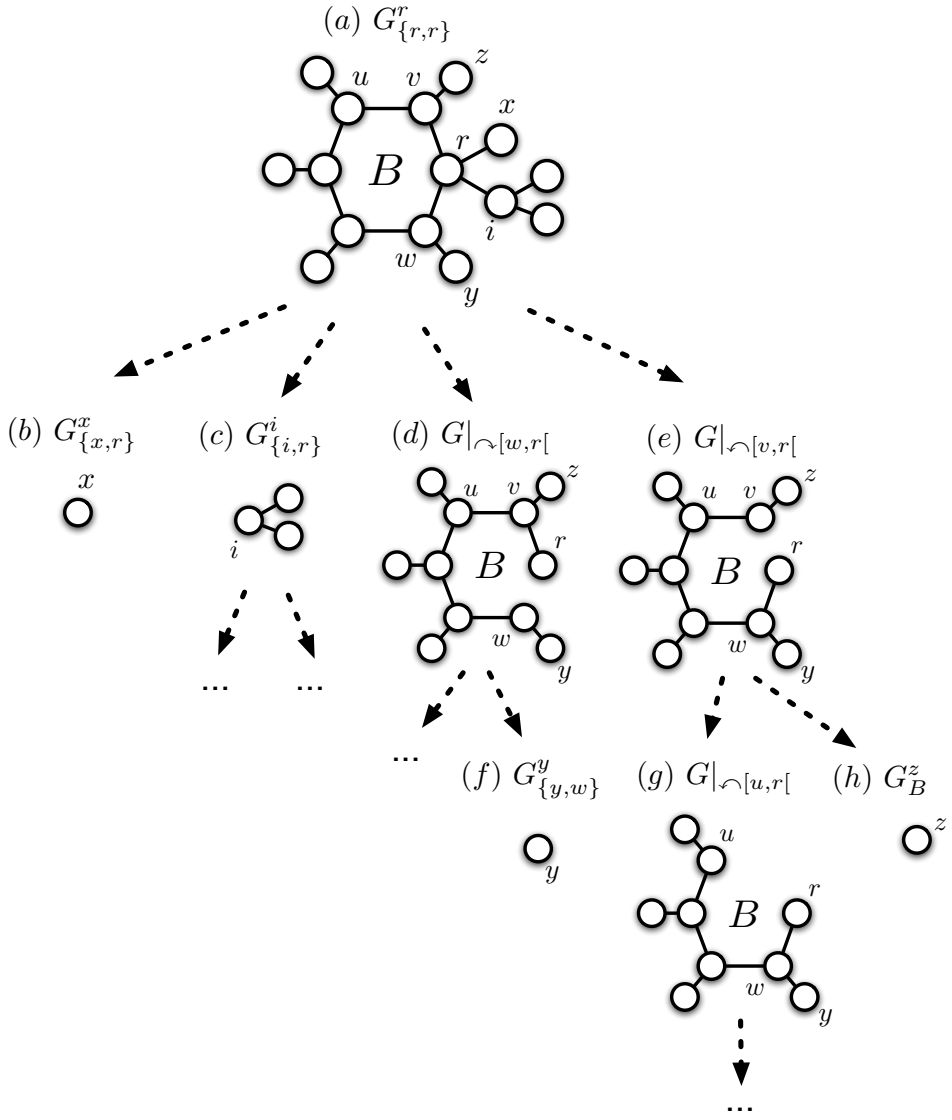


Figure 5.6: (a) A BPS $G^r_{\{r,r\}}$. (b-h) Examples of children. A big dashed arrow from (x) to (y) means that (x) is a parent of (y).

- We can now repeat the above reasoning for $G'^{r'}$. Hence, at some point, a subgraph $G''^{r''}$ can be found such that $r''$ is in the range of $\varphi(S)$ and $G''^{r''}$ satisfies the condition.

**Proposition 5.2** *Let $S$ be the MCS of outerplanar graphs $G$ and $H$ and let $\varphi_G$ and $\varphi_H$ be the respective mappings. Let $\mathcal{R}_G$ be the extended relevant subgraph DAG of $G$ and $\mathcal{R}_H$ the relevant subgraph DAG of $H$. $\mathcal{R}_H$ has a vertex with a subgraph $H's$ such that $s$ is in the range of $\varphi_H$ and $\varphi_H(S)$ is a subgraph of $H's$. Similarly, $\mathcal{R}_G$ has a vertex with a subgraph $G'^r$ such that $r$ is in the range of $\varphi_G$ and $\varphi_G(S)$ is a subgraph of $G'^r$. Moreover, $r$ and $s$ are the image under respectively $\varphi_G$ and $\varphi_H$ of the same vertex in $S$, i.e. $\varphi_G(\varphi_H^{-1}(s)) = r$.*

Proof:

- $H's$ is a relevant subgraph in one of the vertices of $\mathcal{R}_H$ because of Prop. 5.1.

- If we consider one relevant subgraph DAG of $G$, then according to Prop. 5.1, there exists a $G''^{r''}$ such that $r''$ is in the range of $\varphi_G$ and $\varphi_G(S)$ is a subgraph of $G'^{r''}$. However, $\varphi_G(\varphi_H^{-1}(s)) = r''$ is not necessarily satisfied. Because we consider the extended relevant subgraph DAG of $G$, there must exist a $G'^r$ that also satisfies the final condition.

The above proposition ensures that a sufficient number of relevant subgraphs of $G$ and $H$ have been generated in order to find a correct MCS.

Finally, we define for two outerplanar graphs $G$ and $H$ the set $\mathcal{P}(G,H)$, containing the pairwise BPSs and BSSs of $G$ and $H$:

$$\mathcal{P}(G,H) = \{\ (S,T) \mid S \in BPS(G),\ T \in BPS(H)\} \cup$$

$$\{\ (S,T) \mid S \in BSS(G),\ T \in BSS(H)\}$$

We will order the relevant subgraphs in the set $\mathcal{P}(G,H)$ lexicographically according to the function *size*, that is, $(S_1,T_1) \leq (S_2,T_2)$ if $size(S_1) \leq size(S_2)$ or $size(S_1) = size(S_2)$ and $size(T_1) \leq size(T_2)$. This will ensure that, in the bottom-up computation of the MCS, children relevant subgraphs will always be considered before their parents. In the remainder of this chapter, we instantiate the *size* of a graph as the sum of its number of vertices and its number of edges, that is, we chose $w_{\lambda_G(x)} = 1$ for every $x \in V(G) \cup E(G)$.

In Sect. 5.3.1, we have defined a function $\sigma$ from $\mathcal{P}(G,H)$ to $\mathcal{G}_{op}$. We will now follow a dynamic programming approach to compute $\sigma(p)$ for every pair $p$ of $\mathcal{P}(G,H)$.

**Algorithm 5.1** Computing an MCS of two outerplanar graphs $G$ and $H$

**Require:** $G$ and $H$ are outerplanar graphs; $\mathcal{P}(G,H)$ contains all the relevant subgraphs as described in Sect. 5.3.2.

**Ensure:** MCS is an $\text{MCS}_{\sqsubseteq}$ of $G$ and $H$.

1: $MCS \leftarrow G(\emptyset,\emptyset)$
2: **for all** $(p_G,p_H) \in \mathcal{P}(G,H)$ in increasing order
3: **if** $p_G \in BPS(G)$ and $p_H \in BPS(H)$
4: $\sigma(p_G,p_H) \leftarrow$MatchBlockPreservingSubgraph$(p_G,p_H)$
5: **if** $|\sigma(p_G,p_H)| > |MCS|$
6: $MCS \leftarrow \sigma(p_G,p_H)$
7: **else if** $p_G \in BSS(G)$ and $p_H \in BSS(H)$
8: $\sigma(p_G,p_H) \leftarrow$MatchBlockSplittingSubgraph$(p_G,p_H)$
9: **return** $MCS$

### 5.3.3 Bottom-up pairwise MCS computation of the relevant subgraphs

After we have described a systematic way to break the two outerplanar graphs into pieces, i.e. their relevant subgraphs, and determined their parent-child relationships, we now turn to the solving part in our dynamic programming approach: we start by computing MCSs for pairs of small relevant subgraphs and then, we use their solutions to compute new solutions for bigger components until we have found a solution for the original graphs.

In the algorithm, an MCS of two outerplanar graphs $G$ and $H$ is represented as a subgraph of $G$. The bijection $\varphi_G$ then corresponds to the identity function.

As we are using the BBP subgraph isomorphism, we introduce a dummy edge to avoid the edges of blocks becoming bridges when constructing $G|_{o[u,v[}$. We define the extended BSS $G|^*_{o[u,v[}$ to be the BSS $G|_{o[u,v[}$ where an edge $\{u,v\}$ with a special label \$ has been added if $G|_{o[u,v[}$ did not contain an edge between $u$ and $v$ itself (with $w_\$ = 0$). By adding this extra edge, split blocks will only be mapped to other split blocks, and not to bridges or complete blocks.

First let us consider Algorithm 5.1. The main loop of the algorithm iterates over all possible BPS and BSS pairs and makes further calls to the specific matching functions depending on the nature of what should be matched in every step. Note that only solutions for pairs of BPSs can give rise to a resulting MCS (Line 5), since we want to ensure the BBP subgraph isomorphism, requiring that a block in the original graphs is never split (otherwise the remaining block edges become bridges).

We now describe how to match two children of the same type: either a BPS of $G$ and a BPS of $H$ or a BSS of $G$ and a BSS of $H$. The matching of a BPS and a BSS is irrelevant because of the BBP subgraph isomorphism. The key

idea is to consider appropriate combinations of descendants of relevant subgraphs, and to extend their MCSs into an MCS of their respective parents. The dynamic programming approach implies that, given two relevant subgraphs, we always have access to an MCS of all possible pairs of their children.

#### 5.3.3.1 Computing a solution for two BPSs

In order to find an MCS of two BPSs $G^r$ and $H^s$ (whose roots have the same label), we construct a weighted maximal matching between the set of children of $G^r$ and the set of children of $H^s$ using the algorithms of [Munkres, 1957; Shamir and Tsur, 1992]. It is important to note that only combinations of BPSs with BPSs and BSSs with BSSs need to be considered. Moreover, the edges that connect the roots to their respective children must be checked for equality of labels. This will be discussed below.

Consider Algorithm 5.2, as well as an accompanying example in Fig. 5.7. In this figure, we assume that all vertices and edges have the same label, so we do not show them. The function MatchVertex$(r, s)$ takes two vertices $r$ and $s$ as input and returns *true* if both vertices exist and have the same label. Otherwise, it returns *false*. Similarly, the function MatchEdge take two edges as input and returns *true* if both edges have identical labels and *false* otherwise. We use the notation $Ch(G^r)$ for the children of a BPS $G^r$. When combining two graphs $G$ and $G'$, we use the notation $G'' = G \cup G'$ when we actually mean $G''(V \cup V', E \cup E')$.

When an MCS of two BPSs with root vertices $r$ and $s$ has to be computed, first, their respective children are added to the matching (Line 3 of Alg. 5.2). The maximal matching is computed by the max function in Line 4. In Fig. 5.7, every child of $G^r$ and $H^s$ is surrounded by a dashed box.[2] We call a child of $G^r$ $X$ (with root $x$) and a child of $H^s$ $Y$ (with root $y$). Only if the edges $\{r, x\}$ and $\{s, y\}$ have identical labels, the mapping of every pair of children of $G^r$ and $H^s$ gives rise to a weight. This weight is computed by the function ComputeWeight$(r, X, s, Y)$ and corresponds to the MCS (which was computed in an earlier step of the algorithm), together with the vertex $r$ and the edge $\{r, x\}$. Since a BPS is never mapped to a BSS, the weight between such pairs is 0. If the edges $\{r, x\}$ and $\{s, y\}$ do not have identical labels, then the weight is always 0, irrespective of the value of the MCS between the children.

In the case there are multiple maximal matchings possible, the algorithm only returns one possible solution. This ensures that the algorithm runs in polynomial time, but has as drawback that it can miss potentially important alternative MCSs.

After a maximal matching has been computed, the matched edges are returned and the MCSs of the selected children can be expanded by combining the graphs returned by each child of the maximal matching. In Fig. 5.7, the children that are

[2] Because the edge $\{r, r'\}$ does belong to $G^r$ but not to $G|^*_{\curvearrowright[r,r']}$, it is drawn as a dashed line. The same applies to $\{s, s'\}$.

**Algorithm 5.2** Computing an MCS of two block-preserving-subgraphs $G^r$ and $H^s$

**function** MatchBlockPreservingSubgraph($G^r, H^s$)
  **if** MatchVertex($r, s$)
    $M \leftarrow Matching(Ch(G^r), Ch(H^s))$
    **return** $\max_{m \in M} \sum_{(X,Y) \in m}$ ComputeWeight($r, X, s, Y$)
  **else**
    **return** $G(\emptyset, \emptyset)$
**function** ComputeWeight($r, X, s, Y$)
  **if** $X \in BPS(G)$ and $Y \in BPS(H)$
    Let $x$ be the root of $X$ and $y$ the root of $Y$.
    **if** MatchEdge($\{r, x\}, \{s, y\}$)
      **return** $\sigma(X, Y) \cup G(\{r\}, \{\{r, x\}\})$
    **else**
      **return** $G(\emptyset, \emptyset)$
  **else if** $X \in BSS(G)$ and $Y \in BSS(H)$
    Let $X = X|_{o^X[r',r[}$ and $Y = Y|_{o^Y[s',s[}$.
    **if** MatchVertex($r', s'$) and MatchEdge($\{r', r\}, \{s', s\}$)
      **return** $\sigma(X, Y) \cup G(\emptyset, \{\{r, r'\}\})$
    **else**
      **return** $G(\emptyset, \emptyset)$
  **else**
    **return** $G(\emptyset, \emptyset)$

part of the maximal matching are indicated with dashed lines.

#### 5.3.3.2 Computing a solution for two BSSs

In order to find an MCS of two BSSs $G|^*_{o^G[r',r[}$ and $H|^*_{o^H[s',s[}$, splitting a block $B_G$ and a block $B_H$ respectively, with the begin and end points having the same label, we distinguish two cases (see Algorithm 5.3).

As a base case, when $G|^*_{o^G[r',r[}$ or $H|^*_{o^H[s',s[}$ consists of only two vertices in the Hamiltonian path along orientation $o^G$ and $o^H$, we check whether the edge matches. If it does, the resulting MCS consists of this edge and the possible BPS that is connected to $r'$ (of which the MCS has been computed in an earlier step of the algorithm), otherwise the MCS is the empty graph.

In the general case, if both BSSs consist of more two vertices in the Hamiltonian path along orientation $o^G$ and $o^H$, the MCS is the largest graph in a set of graphs $S$. Each graph in this set is obtained by combining two MCSs of children BSSs. All the children BSSs are considered by iterating over the vertices $r''$ on the Hamiltonian

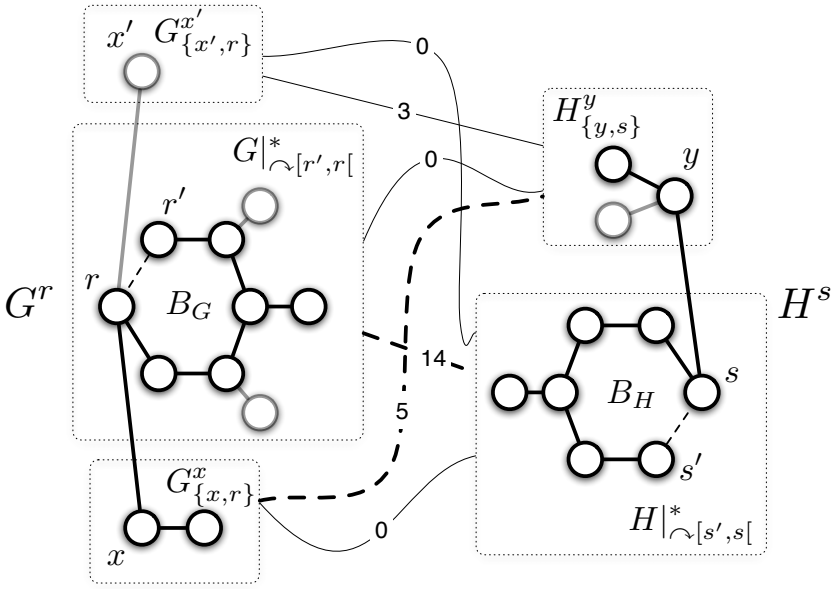


Figure 5.7: An example matching between two BPSs $G^r$ and $H^s$.

path between $r'$ and $r$ according to orientation $o^G$ for which there exists an edge $\{r', r''\}$ and the vertices $s''$ on the Hamiltonian path between $s'$ and $s$ according to orientation $o^H$ for which there exists an edge $\{s', s''\}$. If a value for $r''$ and $s''$ gives rise to two valid BSSs of $G$ and two valid BSSs of $H$, then a new MCS is constructed by connecting the smaller MCS solutions to each other. At the same time, it is checked whether diagonal edges between $r''$ and $f$ or between $r'$ and $r$ exist, and whether they can be mapped to corresponding edges in $H$. Note that also here the algorithm selects one particular combination of BSSs in the case of ties.

This approach is related to the approach of Lingas [1989], who describes an algorithm for subgraph isomorphism between biconnected outerplanar graphs. In Fig. 5.8, there is an example of two BSSs that are mapped.

### 5.3.4 Correctness proof sketch

One can show that the proposed algorithm works correctly by verifying that the trivial cases are solved correctly and that every newly computed MCS of two

**Algorithm 5.3** Computing an MCS of two block-splitting-subgraphs $G|^*_{o^G[r',r[}$ and $H|^*_{o^H[s',s[}$

**Require:** $g_1 = r', g_2, \ldots, g_m = r$ are the vertices of the path $[r', r[$ with orientation $o^G$ of the block $B_G$, $h_1 = s', h_2, \ldots, h_m = s$ are the vertices of the path $[s', s[$ with orientation $o^H$ of the block $B_H$

```
1:  function MatchBlockSplittingSubgraph(G|*_{o^G[r',r[}, H|*_{o^H[s',s[})
2:      if not(MatchVertex(r', r) and MatchVertex(s', s))
3:          return G(∅, ∅)
4:      if o^G[r', r[ or o^H[s', s[ consists of two vertices in the Hamiltonian path
        along orientation o^G or o^H
5:          if MatchEdge({r', r}, {s', s})
6:              return σ(G^{r'}_{B_G}, H^{s'}_{B_H}) ∪ G({r}, {{r', r}})
7:          else
8:              return G(∅, ∅)
9:      else
10:         MCS ← G(∅, ∅)
11:         for all pairs of edges {r', r''} ∈ o^G[r', r[ and {s', s''} ∈ o^G[s', s[ do
12:             if MatchEdge({r', r''}, {s', s''})
13:                 CS ← σ(G|*_{o^G[r'',r[}, H|*_{o^H[s'',s[}) ∪ σ(G^{r'}_{B_G}, H^{s'}_{B_H}) ∪ G(∅, {r', r''})
14:                 if |CS| > |MCS|
15:                     MCS ← CS
16:         return MCS
```

relevant subgraphs is a correct increment of the MCS of their earlier computed children.

**Matching of two BPSs** We first describe the computation of an MCS $S^t$ of two BPSs $G^r$ and $H^s$. We have to prove the following invariant:

$$\sigma(G^r, H^s) = \max\{S^t \mid S^t \sqsubseteq_{\varphi_G} G^r \wedge\ S^t \sqsubseteq_{\varphi_H} H^s\} \tag{5.1}$$

with $\varphi_G(t) = r \wedge\ \varphi_H(t) = s$.

First, if the roots $r$ and $s$ are different, we have defined that there exists no MCS. In this case, it holds that $\lambda_G(r) \neq \lambda_H(s)$, so no isomorphism mappings $\varphi_G$ and $\varphi_H$ can exist such that with $\varphi_G(t) = r \wedge\ \varphi_H(t) = s$. Therefore, the set in Eq. 5.1 is empty, such that the invariant corresponds to the empty MCS.

Second, when the roots are equal, that is, $\lambda_G(r) = \lambda_H(s)$, we consider two cases:

- **Base case** When either $G^r$ or $H^s$ consists of a single vertex, no maximal matching can be created, since a single vertex cannot have children. In this

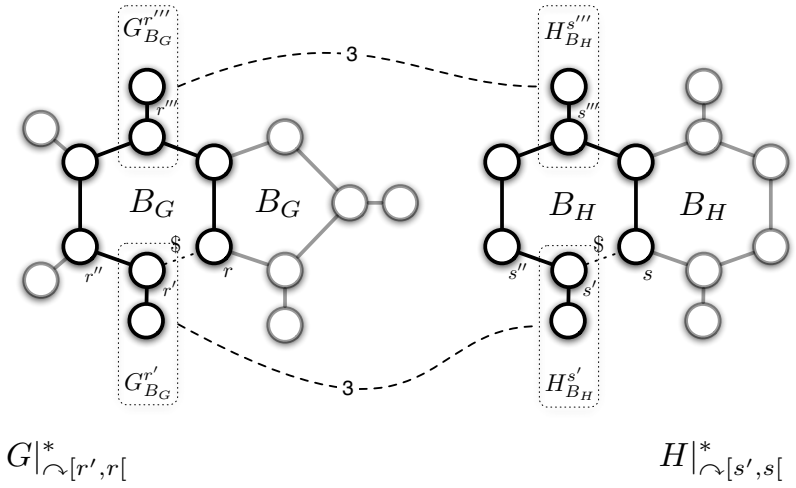


Figure 5.8: An example matching between two BSSs $G|^*_{o^G[r',r[}$ and $H|^*_{o^H[s',s[}$.

case, $S^t$ is a graph consisting of one single vertex labeled $\lambda_G(r)$. Since $r$ and $s$ are the roots of $G^r$ and $H^s$, it is trivial to show that $S^t$ is subgraph isomorphic to $G^r$ and $H^s$, that is, the following holds:

$$S^t \sqsubseteq_{\varphi_G} G^r \wedge \ S^t \sqsubseteq_{\varphi_H} H^s.$$

Now we just need to show that

$$\varphi_G(t) = r \wedge \ \varphi_H(t) = s.$$

However, since $\lambda_G(r) = \lambda_H(s) = \lambda_G(t)$, we can show that such a mapping exists. Moreover, since there is only one possible $S^t$ consisting of one single vertex labeled $\lambda_G(r)$, it is automatically the largest one.

- **Induction step** In order to prove the correctness in the general case, we assume a matching between the children of $G^r$ and $H^s$ and assume without loss of generality that only one pair of BPS children ($G^x$ and $H^y$) has been selected in the matching. Because of the induction hypothesis, we can also assume to have a solution for an MCS $T^u$ between $G^x$ and $H^y$, that is, we know that:

$$T^u \sqsubseteq_{\varphi_G} G^x \wedge \ T^u \sqsubseteq_{\varphi_H} H^y \wedge \ \varphi_G(u) = x \wedge \ \varphi_H(u) = y.$$

Now we need to proof that the MCS $S^t$ is a correct increment of the MCS

$T^u$, or,

$$S^t \sqsubseteq_{\varphi_G} G^r \wedge\ S^t \sqsubseteq_{\varphi_H} H^s \wedge\ \varphi_G(t) = r \wedge\ \varphi_H(t) = s, \tag{5.2}$$

and that it is maximal. We start by proving that $S^t \sqsubseteq_{\varphi_G} G^r$. First of all, the BPS $G^r$ is exactly the same as its child BPS $G^x$, except for one additional vertex $(r)$ and edge $(\{r, x\})$. $S^t$ can thus have at most one vertex and edge more than $T^u$. Since we have assumed that the root $r$ can be mapped, we only need to consider the possible matching of $\{r, x\}$. However, if this edge cannot be mapped, then the child $G^x$ would not have been considered in the maximal matching. We can thus conclude that it can be mapped, and thus $S^t \sqsubseteq_{\varphi_G} G^r$.

The proof for $S^t \sqsubseteq_{\varphi_H} H^s$ is identical.

Now we only have to prove that $\varphi_G(t) = r \wedge\ \varphi_H(t) = s$. However, again because we have assumed equal root vertices, we can choose such an isomorphism mapping.

We still have to proof that $S^t$ is maximal, that is, there exists no other graph that also fulfils the requirements in Eq. 5.2 and that is strictly larger. However, since $G^r$ is only one vertex and one edge larger than $G^x$, and since we have extended the MCS $T^u$ with exactly one vertex and one edge to reach $S^t$, we know that there cannot exist an $S^t$ that is strictly larger.

A similar proof can be constructed in the case of two BSS children that are being matched. $G^r$ and $H^s$ then have children $G|^*_{o^G[r',r[}$ and $H|^*_{o^H[s',s[}$, respectively. An MCS between the children has been computed in a previous step, and only the mapping of the edges $\{r, r'\}$ and $\{s, s'\}$ needs to be checked. If the edges have equal labels, their weight in the maximal matching is equal to the size of the MCS together with the edge $\{r, r'\}$. If not, their weight in the maximal matching is 0.

**Matching of two BSSs** We now turn to the computation of an MCS $S|^*_{o^S[t',t[}$ of two BSSs $G|^*_{o^G[r',r[}$ and $H|^*_{o^H[s',s[}$, splitting a block $B_G$ and $B_H$, respectively. We have to prove the following invariant:

$$\sigma(G|^*_{o^G[r',r[}, H|^*_{o^H[s',s[}) = \max_{S \in \rho(G|^*_{o^G[r',r[}, H|^*_{o^H[s',s[})} S. \tag{5.3}$$

with $\rho(G|^*_{o^G[r',r[}, H|^*_{o^H[s',s[})$ the set of all graphs $S|^*_{o^S[t',t[}$ for which $S|^*_{o^S[t',t[} \sqsubseteq_{\varphi_G} G|^*_{o^G[r',r[}$ and $S|^*_{o^S[t',t[} \sqsubseteq_{\varphi_H} H|^*_{o^H[s',s[}$ and for which there are two BBP subgraph isomorphism mappings $\varphi_G : S|^*_{o^S[t',t[} \to G|^*_{o^G[r',r[}$ and $\varphi_H : S|^*_{o^S[t',t[} \to H|^*_{o^H[s',s[}$ such that $\varphi_G(t') = r'$, $\varphi_G(t) = r$, $\varphi_H(t') = s'$ and $\varphi_H(t) = s$.

First, if the endpoints $r$ and $s'$ or $r$ and $s$ are different, we have defined that there exists no MCS. In this case, the set $\rho$ is empty, such that the invariant corresponds to the empty MCS.

Second, if the endpoints are equal, that is $\lambda_G(r') = \lambda_H(s')$ and $\lambda_G(r) = \lambda_H(s)$, we consider two cases:

- **Base case** When $G|^*_{o^G[r',r[}$ or $H|^*_{o^H[s',s[}$ consists of two vertices of the Hamiltonian path, then it is checked if $\lambda_G(\{r', r\}) = \lambda_H(\{s', s\})$. If this is false, $\rho$ is again empty, and there is no MCS. Otherwise, $S$ is the result of combining the two vertices $r'$ and $r$ and their corresponding edge $\{r', r\}$, together with the MCS $M$ of the BPSs $G^{r'}_{B_G}$ and $H^{s'}_{B_H}$. Because of the induction hypothesis we can assume that $M$ is maximal, and since the new MCS has been extended with the maximal amount of vertices and edges, $S$ is again maximal.

- **Induction step** We assume there is an MCS $S'|^*_{o^{S'}[t'',t[}$ of BSSs $G|^*_{o^G[r'',r[}$ and $H|^*_{o^H[s'',s[}$. It is straightforward to show that, if the edges $\{r', r''\}$ and $\{s', s''\}$ can be matched, that is, $\lambda_G(\{r', r''\}) = \lambda_H(\{s', s''\})$, the MCS $S'$ can be correctly extended to $S$ by adding the edge $\{r', r''\}$ together with $\sigma(G^{r'}_{B_G}, H^{s'}_{B_H})$ (see Fig. 5.8).

### 5.3.5 Time complexity

In this section, we outline the proof for the following theorem:

**Theorem 5.1** *The proposed algorithm computing an MCS under BBP subgraph isomorphism of two given outerplanar graphs in time* $O(|V(G)|^{5/2}.|V(H)|^{5/2})$.

Proof sketch: The result follows from counting the number of relevant subgraphs of the outerplanar graphs to consider and using known bounds on the running times of the algorithms used in the dynamic programming step.

Consider the algorithm as explained in Sect. 5.3.3. We start with an optimisation note. If ComputeWeight is called multiple times with the same arguments, we can avoid doing the computation again by remembering the result.

Let us first consider the time spent in MatchBlockPreservingSubgraph (Alg. 5.2). A call to ComputeWeight can be performed in constant time except if $x$ and $y$ refer to blocks. Line **??** is executed at most once for every ordered pair $(r, r')$ and $(s, s')$, so for function ComputeWeight we can account $O(|V(G)|.|V(H)|)$ time.

There are at most $O(|V(G)|.\Delta_G)$ elements in $BPS(G)$ where $\Delta_G$ is the maximal degree of a vertex of $G$. MatchBlockPreservingSubgraph is thus called $O(|V(G)|.\Delta_G.|V(H)|.\Delta_H)$ times. Every call to MatchBlockPreservingSubgraph costs (except for the time spent in ComputeWeight discussed above) at most the time to solve a weighted maximal matching problem, which can happen in cubic time [Munkres, 1957]. Therefore, a rough bound on the time spent in MatchBlockPreservingSubgraph is $O(|V(G)|^{5/2}.\Delta_G.|V(H)|^{5/2}.\Delta_H)$.

In function MatchBlockSplittingSubgraph (Alg. 5.3), the for loop is executed at most four times for every value of $v_b^G$, $v_m^G$, $v_e^G$, $v_b^H$, $v_m^H$ and $v_e^H$. Therefore, the time spent in MatchBlockSplittingSubgraph is bounded by $O(|V(G)|^3.|V(H)|^3)$.

The above arguments show that there is an algorithm running in time bounded by a polynomial of degree 5.

## 5.4 Related work

There exist a number of approaches that compute MCSs of graphs and there are a lot of differences in the way they handle the NP-completeness of the problem. For example, certain algorithms solve the MCS problem optimally, while others do it approximately. Moreover, there exist slightly different notions of an MCS e.g., whether it is connected or whether it is subgraph-induced. We will first discuss these different notions.

First, the computed MCS can be connected or unconnected. When computing an unconnected MCS, a trade-off has to be made between the number of connected fragments in the MCS and its usefulness as a pattern. If too many connected components are allowed, the pattern becomes too general. If the number is too small, the pattern may be too specific. For example, when two molecules share two benzene rings but differ with respect to the connection between the rings, only one of these rings can appear in a connected MCS. Despite this potential drawback, we focus on finding connected MCSs here.

A second distinction is made between the maximum common induced subgraph (MCIS) and the maximum common edge subgraph (MCES). The MCIS requires the common subgraph to be induced, in the sense that if two vertices in the first graph are linked by an edge, they are mapped to two vertices in the second graph that are linked by an edge as well.[3] The MCES, which does not have this requirement, simply tries to maximise the number of edges in the common subgraph. Chemists have argued that the MCES more adequately expresses the notion of chemical similarity than does the MCIS, since the bonded interactions between atoms are most responsible for the perceived molecular activity [Raymond and Willett, 2002b]. An MCS that is computed between outerplanar graphs under the BBP subgraph isomorphism corresponds to neither of these notions, but tries to maximise the edges given that only bridges are mapped to bridges and block edges to block edges.

Here we also introduce a third distinction, which is the kind of subgraph isomorphism used in the matching. Usually, the general subgraph isomorphism is used to compute the MCS, while we propose to use the BBP subgraph isomorphism instead.

[3]The definition of an induced subgraph $G$ of a graph $H$ requires that for every $u, v \in V(G)$: $\{u, v\} \in E(G)$ iff $\{\varphi(u), \varphi(v)\} \in E(H)$.

An overview of MCS algorithms has been given by Raymond and Willett [2002b] and Conte et al. [2004]. McGregor [1982] proposes a simple backtracking strategy that uses various heuristics in order to reduce the number of backtrackings. Akutsu [1993] proposes a polynomial algorithm for computing the connected MCES of "almost trees of bounded degree" under the general subgraph isomorphism. This is another class of graphs which is suited for the representation of molecules. Koch [2001] introduces an algorithm for enumerating all connected MCESs in two graphs, based on a transformation of the two graphs into an association graph. The MCS problem is then transformed into a clique detection problem. Raymond et al. [2002] use the same problem transformation and propose an exact multi-step algorithm which defines a similarity based on computing the unconnected MCES. The problem still remains theoretically NP-complete, but their algorithm makes use of advanced heuristics to reduce the number of matchings required.

Cao et al. [2008] propose an improved backtracking algorithm that is specialized for molecules and works directly on the chemical graph. Because of branch-and-bound heuristics, they can drastically reduce the search space and obtain an efficient algorithm for unconnected MCIS computation. They show that their MCS-based similarity measure outperforms traditional descriptor-based and topology-based similarity measures. In the next section, we will perform a runtime comparison with this algorithm.

Instead of using heuristics, an alternative approach to reduce the complexity could be to consider common substructures which can be computed more easily, such as multisets of common vertex labels [Karunaratne and Boström, 2006]. However, an important drawback is that the more complex shared substructures are not taken into account.

Our algorithm is different from the other approaches in the sense that it computes a connected MCS that fulfils the requirements of the BBP subgraph isomorphism in an exact way. The algorithm works directly on the chemical graph and does not require an advanced graph-theoretical problem transformation.

## 5.5 Experiments

In this section, we compare our polynomial MCS algorithm against the algorithm of Cao et al. [2008] in terms of efficiency. This algorithm computes an unconnected MCS under the general subgraph isomorphism (and is not bounded by the restriction to outerplanar graphs). Cao et al. [2008] make use of new heuristics in their algorithm to drastically reduce the search space compared to other MCS algorithms, so this algorithm should be one of the fastest available implementations of an algorithm that computes MCSs under the general subgraph isomorphism. To the best of our knowledge, other implementations of MCS algorithms are not freely available in the public domain.

### 5.5.1 Dataset

The NCI dataset has been made publicly available by the National Cancer Institute and provides screening results for the ability of more than 70,000 compounds to suppress or inhibit the growth of a panel of 60 human tumour cell lines. Each of the compounds is described by its 2D representation, that is, the structure of their atoms and bonds. We use graphs to represent the molecules, in which the vertices are labeled with general atom types (e.g., carbon, nitrogen) and the edges are labeled single, double, triple, amide or aromatic. Hydrogen atoms are dropped. The NCI dataset can be downloaded from `http://cactus.nci.nih.gov`.

We have selected a subset of 50 molecules of different sizes from the NCI dataset, with the number of vertices ranging from 5 to 99 (average: 45.7) and the number of edges ranging for 4 to 107 (average: 49.4).

### 5.5.2 Method

For the set of 50 graphs, there are 1225 possible combinations of two graphs. For each of them, we will compute a connected $\text{MCS}_{\sqsubseteq}$ with the algorithm we proposed in Sect. 5.3.3 and an unconnected $\text{MCS}_{\preceq}$ with the algorithm of Cao et al. [2008] and measure runtimes. From now on, we will refer to these algorithms as $\text{MCS}_{\sqsubseteq}$ and $\text{MCS}_{\preceq}$.

$\text{MCS}_{\sqsubseteq}$ can be downloaded at `http://www.cs.kuleuven.be/~dtai/PMCSFG`, while $\text{MCS}_{\preceq}$, part of a package called ChemMiner, can be downloaded at `http://biowb.ucr.edu/ChemMineV2/help/mcs.html`. All experiments were run on an Intel Core2Quad 2.33 GHz processor. A time-out of 24 hours per MCS computation was enforced.

### 5.5.3 Results

A comparison of the runtimes can be found in Fig. 5.9. Notice that a logarithmic scale is used. The figure clearly shows the exponential behaviour of $\text{MCS}_{\preceq}$. For 18 of the 1225 MCS computations, $\text{MCS}_{\preceq}$ timed out at 24 hours; we do not show the runtimes of $\text{MCS}_{\sqsubseteq}$ on these 18 comparisons either. We also report the order statistics of the computation time for both algorithms in Table 5.1. These show that $\text{MCS}_{\sqsubseteq}$ computes an MCS in less than 0.713 seconds in 95% of the cases, while $\text{MCS}_{\preceq}$ needs 1643 seconds for this. However, in 204 cases, $\text{MCS}_{\preceq}$ was faster than $\text{MCS}_{\sqsubseteq}$, which indicates that both algorithms are competitive when computing MCSs for smaller molecules. The graph in Fig. 5.9 confirms this.

While comparing these runtimes, we need to keep in mind that, next to the difference in subgraph isomorphism, there is also another difference between the MCS algorithms. $\text{MCS}_{\preceq}$ computes an unconnected MCS, while the MCS of $\text{MCS}_{\sqsubseteq}$ is connected. This factor also contributes to the improved runtimes of $\text{MCS}_{\sqsubseteq}$ and

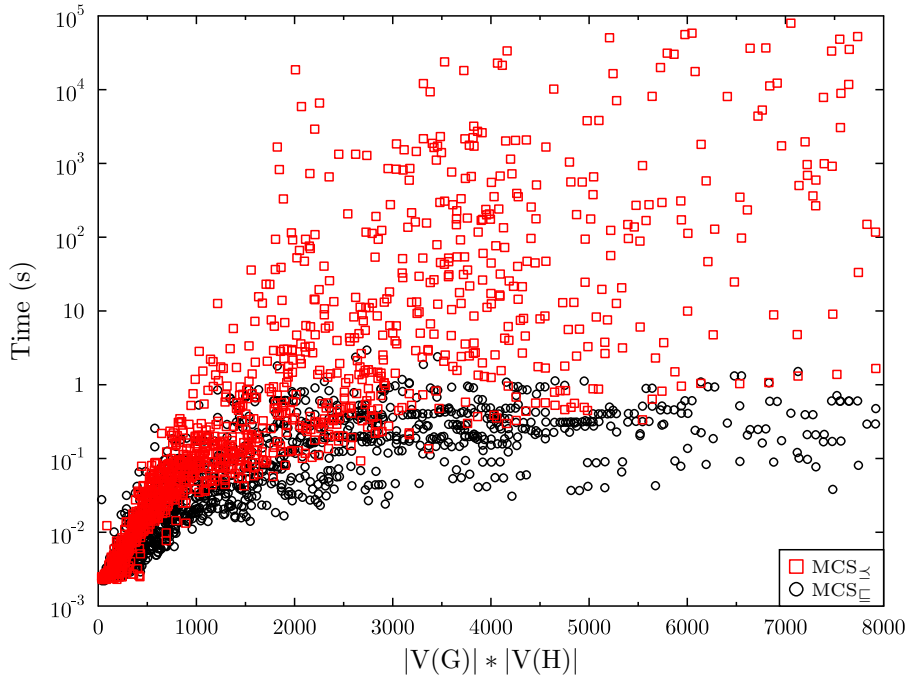


Figure 5.9: Runtimes of MCS$_{\sqsubseteq}$ and MCS$_{\preceq}$ (in seconds).

it would be interesting to be able to measure the impact of these two restrictions separately.

**Conclusion** The results of the experimental comparison between MCS$_{\preceq}$ and MCS$_{\sqsubseteq}$ show that MCS$_{\sqsubseteq}$ outperforms MCS$_{\preceq}$ in terms of efficiency. This can be explained by four restrictions that are imposed by MCS$_{\sqsubseteq}$: it works with outerplanar graphs, makes use of the BBP subgraph isomorphism, computes the connected MCS and selects a single random one in the case there are multiple MCSs.

## 5.6 Conclusions

In this chapter, we have introduced an algorithm that computes an MCS of two outerplanar graphs under the BBP subgraph isomorphism. We have proved its correctness and shown that it runs in polynomial time, both theoretically and empirically.

Table 5.1: Distribution of runtimes of the two MCS algorithms for 1207 comparisons (in seconds).

| | $MCS_{\preceq}$ | $MCS_{\sqsubseteq}$ |
|---|---|---|
| Average | 805.668 ± 5177.040 | 0.174 ± 0.282 |
| Minimum | 0.002 | 0.002 |
| O.05 | 0.003 | 0.003 |
| O.25 | 0.025 | 0.012 |
| O.5 | 0.161 | 0.049 |
| O.75 | 3.590 | 0.238 |
| O.95 | 1643.850 | 0.713 |
| Maximum | 79625.900 | 2.951 |

The polynomial time complexity is realised by four restrictions. First, the algorithm only works on outerplanar graphs. Most molecular graphs, which are of most interest to us, are outerplanar. Second, the algorithm uses the BBP subgraph isomorphism as matching operator. Again, in the chemical context, this seems to be a sensible choice. Third, we only consider connected MCSs. Fourth, in the case there are multiple possible MCSs, we only select one randomly.

Finally, we have shown that our algorithm outperforms a state-of-the-art MCS algorithm that computes an unconnected MCS under the general subgraph isomorphism.

# Chapter 6

# An Efficiently Computable Metric for Outerplanar Graphs

## 6.1 Introduction

In this chapter, we will present a metric that can be used on outerplanar graphs. A metric is a function which defines a distance between elements of a set. Metrics are important components of several machine learning methods. For example, they are used in classification techniques such as the $k$-nearest-neighbours algorithm, where examples are classified based on the class values of their nearest neighbours in the training set. They can also be used to perform clustering, where the task is to group similar examples together and where some kind of distance measure is used to define similarity. Recently, there has been an increased interest in metrics that express a similarity between *structured* objects. This is relevant for multiple application domains, including drug discovery [Ceroni et al., 2007], image recognition and computer vision [Shearer et al., 2001], and geographical databases [Li et al., 2005].

As mentioned in the outline of part II, the application on which we focus is the learning of structure-activity relationships. Since it is widely known that molecules with a similar structure tend to have the same function [Johnson and Maggiora, 1990], structural similarity search among small molecules is an important aspect of *in silico* drug development. The task then comes down to finding an appropriate similarity measure between molecules. Such a structural similarity measure should ideally fulfil two requirements: (1) it should be efficiently computable, which is important when analysing large molecular databases, and (2) the notion of similarity

should discriminate well between molecules w.r.t. the activity at interest. Finding such similarity measures is one of the current challenges in chemoinformatics [Deshpande et al., 2005; Ceroni et al., 2007].

However, similarity measures between graphs that aim at using all available structural information often involve the matching of subgraphs or other combinatorial operations trying to align graphs optimally. For this reason, typical approaches have resorted to a transformation of molecules into vectors and compute a similarity between those vectors. In such transformations, either information is lost or the size of the resulting representation can grow exponentially [De Raedt, 1998].

Instead of resorting to a vector transformation to avoid the computational complexity, we will make use of an efficient algorithm for graphs. More specifically, we will use the notion of a maximum common subgraph (MCS) to define a similarity measure between molecules. The idea behind this is that an MCS of two molecular graphs may reflect shared properties that relate to the molecular activity. In Chapter 5, we have introduced a polynomial algorithm that computes an MCS under the BBP subgraph isomorphism. In this chapter, we will show that the metric based on this algorithm has two important properties: (1) it is computable in polynomial time, and (2) as it reflects the size of the MCS, it has an intuitive meaning making results of methods such as instance-based learning interpretable. Whether the BBP subgraph isomorphism provides the right level of abstraction in order to discriminate between molecules with a different activity and how this influences the predictive performance of classification methods using the BBP subgraph isomorphism as matching operator, is an important issue that will be investigated in this chapter.

The contributions of this chapter are the following:

- We introduce an efficiently computable metric on outerplanar graphs based on the maximum common subgraph computed under the BBP subgraph isomorphism.
- We compare the metric against several other metrics, one of which is based on the MCS computed under the general subgraph isomorphism, in terms of predictive performance.
- We investigate the usefulness of the BBP subgraph isomorphism in general by performing additional experiments in the context of frequent subgraph mining.

The contents of this chapter are the result of joint work with Jan Ramon, Maurice Bruynooghe and Hendrik Blockeel, and was partially published in [Schietgat et al., 2008].

The chapter is organised as follows. We start by discussing related work in Sect. 6.2. In Sect. 6.3, we present the metric for outerplanar graphs. In Sect. 6.4, we answer three experimental questions and finally, in Sect. 6.5, we conclude.

## 6.2 Related work

Determining quantitative structure-activity relationships (QSAR) is the process where chemical structure is quantitatively correlated with a biological or chemical activity. Usually, a function is learned that takes as input a number of physicochemical and structural properties of a molecule and then returns its predicted activity. A number of learning approaches have been proposed for QSAR. Depending on the representation they use, we will organise them in several categories.

Although it is known that better performance can be obtained using additional 3D information such as force field calculations [Ceroni et al., 2007], we will concentrate on 2D-QSAR methods, that is, methods that only take into account the structural arrangement of atoms and bonds.

### 6.2.1 Descriptor-based methods

The first methods for QSAR were *descriptor-based*: using various measurable physicochemical properties (such as polarity or hydrophobicity) of a molecule as descriptors, the molecule can be transformed into a vector of real numbers, each number representing a specific property [Hansch et al., 1962]. Then, a number of propositional similarity measures (e.g., the Tanimoto coefficient [Willett, 2006]) and machine learning methods can be applied.

There are two main difficulties with this approach: (1) domain knowledge is required to select the proper descriptors, and (2) information about the molecular structure is lost. As there is a common agreement that the structure is important when developing similarity measures, there has been an increased interest in relational learning methods.

### 6.2.2 Relational methods

Because relational methods take structural information directly into account, these methods are often referred to as *topology-based* approaches. One framework in which relational approaches have been developed is that of inductive logic programming (ILP), where the molecules and models for them are described with first-order logic. In this domain, a number of algorithms have been proposed that obtain excellent results on several QSAR tasks [King et al., 1996; Helma et al., 2004]. However, because of computational complexity issues, it is still a challenge for ILP algorithms to cope with large datasets.

Actually, the ILP representation is much more general than what is needed to model molecules and to discover substructures. Therefore, a number of methods aim at achieving better performance by building systems that use special-purpose data structures for representing graphs and provide well-chosen primitives for manipulating them (e.g., [Yan and Han, 2002; Nijssen and Kok, 2004; Chi et al.,

2005]). In this way, they transform the problem of binding chemical substructures into that of finding subgraphs in a graph. Still, finding subgraphs requires expensive subgraph isomorphism matchings.

A number of such specialised graph algorithms have been developed to compute similarities between graphs [Raymond and Willett, 2002b; Conte et al., 2004], such as the MCS algorithms that were discussed in Sect. 5.4. De Raedt and Ramon [2009] show that the notion of a maximum common subgraph (or a minimally general generalisation in relational learning terms) can be used to construct a distance measure.

Furthermore, kernels can be viewed as a kind of similarity measure as well. Following the increased emphasis on structure, several kernel functions have been developed that work on graphs directly [Horváth et al., 2004; Gärtner, 2005; Swamidass et al., 2005; Ceroni et al., 2007; Shervashidze and Borgwardt, 2009]. Again, the main problem here is to select kernel functions that can capture the molecule's activity while still remaining efficient to compute. Many types of graph kernels correspond to some implicit feature space where features are generated for certain subgraphs or graph properties. The kernel then counts how many subgraphs two examples have in common. The subgraphs are typically defined in such a way that they can be enumerated implicitly and such that the counting procedure can be done efficiently (e.g., via dynamic programming procedures). For example, Ceroni et al. [2007] have introduced the weighted decomposition kernel (WDK), which obtains state-of-the-art results when used for the classification of molecules. It is based on a decomposition of the molecule in a selector (a single vertex) and a context (a fixed-radius subgraph surrounding the selector). By selecting an appropriate kernel for these structures, the computation of the WDK kernel remains feasible.

### 6.2.3 Propositionalisation methods

To avoid having to deal with subgraph isomorphisms during the learning phase, a popular approach in relational learning is to propositionalise the graph-based representation into an attribute-valued representation [De Raedt, 2008]. In this context, the learning process involves two separate steps: first, the complete dataset of graphs is searched for subgraphs and then, each example is represented as a bit-vector, which is often called *fingerprint*, that encodes the occurrences of these subgraphs in the example. The difference with the above mentioned descriptor-based methods is that the vectors here encode topological features instead of numerical ones, trying to minimise the loss of structural information. Since the graph miner automatically selects the "most interesting" features, this approach solves the first difficulty mentioned above, while the second difficulty is transformed into selecting a suitable criterion that finds the most interesting features, i.e. the patterns that describe the molecule's topology best in order to discriminate molecules w.r.t. their activity.

To this end, various techniques have been used to generate features of interest [Kramer et al., 2001]. In the chemoinformatics context, there are two important approaches: either the patterns are generated exhaustively, or some kind of constraint is used to select the most interesting patterns.

The current state-of-the-art methods towards feature construction for molecules generate all sequences [Willett, 2006] or subgraphs [Wale et al., 2008] of size up to $k$ that occur in at least one molecule. Because even for small values of $k$, this rapidly leads to vast numbers of features, the generated features are typically compressed using some kind of hashing of the occurrences of the paths onto a fixed-lengh vector. Because of its size, the resulting pattern set is hard to interpret, and the use of hashing methods only aggravates this problem.

In order to limit the size of the pattern set, a number of constraint-based graph mining methods have been proposed, where constraints on the subgraphs of interest are formulated. A wide variety of different constraints has been considered, such as frequency [Deshpande et al., 2005; Wale et al., 2008], statistical significance measures such as $\chi^2$ [Bringmann et al., 2006; He and Singh, 2006], syntactical constraints [Kramer et al., 2001], randomisation [Chaoji et al., 2008] or a combination of those [Maunz et al., 2009]. In these types of approaches, one typically performs a complete search. This leads one to finding all patterns satisfying the constraints.

## 6.3 A metric for outerplanar graphs

In this section, we show how we can use the concept of an MCS to define a metric on outerplanar graphs.

### 6.3.1 Definition

A pseudo-metric on a set $\Omega$ is a function $d : \Omega \times \Omega \to \mathbb{R}_+$ that fulfils three requirements:

- *reflexivity*: $\forall x \in \Omega : d(x, x) = 0$,
- *symmetry*: $\forall x, y \in \Omega : d(x, y) = d(y, x)$,
- *triangle inequality*: $\forall x, y, z \in \Omega : d(x, z) \leq d(x, y) + d(y, z)$.

If it also holds that $\forall x, y \in \Omega : d(x, y) = 0 \Rightarrow x = y$, then $d$ is called a metric.

### 6.3.2 Using the maximum common subgraph notion to construct a metric

The goal of this chapter is to develop a metric on $\mathcal{G}_{op}^{\equiv}$. Given two graphs $G$ and $H$, Bunke and Shearer [1998] proposed a distance function on graphs based on the

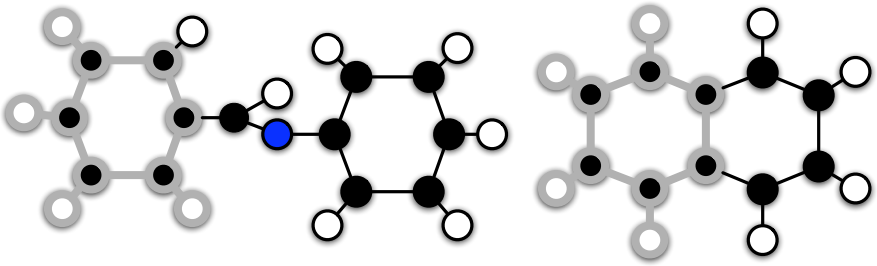

Figure 6.1: An MCS computed under the block-and-bridge-preserving subgraph isomorphism.

maximum common subgraph:

$$d_{bs}(G,H) = 1 - \frac{|MCS(G,H)|}{\max(|G|,|H|)}, \tag{6.1}$$

with $|G|$ and $|H|$ equal to the number of vertices in $G$ and $H$, respectively. They proved that $d_{bs}$ is a metric. It is possible to extend this proof to the more general case where $|G|$ and $|H|$ are determined by the function *size*. Some variants with similar properties and performances are reviewed by Raymond and Willett [2002a].

By restricting the metric to outerplanar graphs, we can use the polynomial MCS algorithm introduced in Sect. 5.3.

**Example 6.1** *Consider the example in Fig. 6.1, where an* $MCS_{\sqsubseteq}$ *is shown between two molecular graphs. We define the size of a graph as its number of vertices here. Here, the size of the MCS is 10, while the largest graph has 25 vertices. According to the above formula, the similarity between the two graphs is then* $1 - \frac{10}{\max(18,25)} = 0.6$.

## 6.4 Experiments

In this section, we want to answer the following questions:

**Q1** How does the predictive performance of the proposed metric based on the $\text{MCS}_{\sqsubseteq}$ compare to the same metric based on the $\text{MCS}_{\preceq}$?

**Q2** What is the relationship between the best MCS-based metric and state-of-the-art metrics?

**Q3** What is the usefulness of the BBP subgraph isomorphism as matching operator in general?

In order to answer Q1, we will compare the metric based on the $\text{MCS}_{\sqsubseteq}$, computed by the algorithm we proposed in Sect. 5.3, to the metric based on the $\text{MCS}_{\preceq}$, computed by the algorithm of Cao et al. [2008]. This algorithm was discussed in Sect. 5.4. In order to answer Q2, we will compare the best MCS-based metric to a metric based on 2D fingerprints [Willett, 2006] and a metric based on the WDK kernel [Ceroni et al., 2007]. We will evaluate all metrics in an instance-based learning setting (IBL).

Although the gains in efficiency that can be obtained by using the BBP matching operator have been studied in detail by Horváth et al. [2006], the impact of the BBP matching operator on predictive performance has never been investigated before. Evaluating the BBP matching operator is interesting in its own, and these results will give us an answer to Q3. To this end, we will compare the different matching operators when used in support vector machines (SVMs) [Joachims, 2002], a method that has become very popular for the classification of molecules (e.g., [Swamidass et al., 2005; Deshpande et al., 2005; Ceroni et al., 2007]). Rather than obtaining state-of-the-art results, the purpose of this experiment is investigating whether the BBP subgraph isomorphism has a larger predictive power compared to the general one.

### 6.4.1 Datasets

The Developmental Therapeutics Program (DTP) at the U.S. National Cancer Institute (NCI) has checked a large number of compounds for evidence of the ability to inhibit the growth of human tumor cell lines.[1] The target values correspond to the parameter $\text{GI}_{50}$, the concentration that causes 50% growth inhibition.

In total, there are 60 datasets, each corresponding to a particular cell line. We use the roughly balanced datasets of Swamidass et al. [2005], which have become a popular benchmark for QSAR algorithm research, and are often referred to as NCI60. Each cell line has inhibition data on about 3500 compounds, which defines a binary classification problem. There are 3910 distinct molecules over all datasets. The average number of vertices is 23, while the average number of edges is 25.

Each molecule is described in the Tripos Sybyl MOL2 format[2], from which we extract a graph in which the vertices are labeled with general atom types (e.g., N, C) and the edges are labeled single, double, triple, amide or aromatic. Hydrogen atoms are dropped.

It is important to note that, in order to compare with the BBP-based methods, we have removed the non-outerplanar examples (≈9.69%) from these datasets.

[1] `http://dtp.nci.nih.gov/docs/cancer/cancer_data.html`
[2] `http://www.tripos.com/data/support/mol2.pdf`

### 6.4.2 Method

**IBL-based classifiers** First, we define the size of a graph as its number of vertices here, that is, $w_{\lambda_G(x)}$ is set to 1 if $x \in V(G)$, 0 if $x \in E(G)$. We call the metric computed under the BBP subgraph isomorphism $d_{\sqsubseteq}$ and the metric computed under the general subgraph isomorphism $d_{\preceq}$. Next to the MCS-based metrics, we construct a metric based on 2D fingerprints and a metric based on the WDK kernel.

For each molecule, we construct a 1024 FP2-fingerprint using OpenBabel v2.1.1[3] and we define a metric on these fingerprints using the Tanimoto coefficient, which is still considered among chemists to produce state-of-the-art results for virtual screening [Willett, 2006]. We call this metric FP2. Although the WDK kernel was not intended to be used in this way, for reasons of comparison we defined a metric according to the following formula: $d^2(x,y) = \kappa(x,x) - 2\kappa(x,y) + \kappa(y,y)$, with $\kappa$ the WDK kernel function. We call this metric WDK.

To compare the predictive performance of the different metrics, we use $k$-nearest neighbour classification (kNN): in order to classify a given molecule, we select the $k$ neighbour molecules that are closest according to the metric (we choose $k = 11$, which resulted in an optimal AUROC for all metrics). For each molecule, we obtain a prediction equal to the percentage of positive votes of its neighbors. In this way, we can rank the predictions and compute the area under the ROC curve (AUROC). We use leave-one-out cross-validation.

**SVM-based classifiers** For the experiments involving SVMs, we transform molecules into bit-vectors using two approaches. The first approach mines frequent subgraphs under the BBP subgraph isomorphism using the FOG algorithm presented in Horváth et al. [2006]. The second approach mines frequent subgraphs under the general subgraph isomorphism, using an efficient implementation [Bringmann et al., 2006] of the gSpan algorithm [Yan and Han, 2002]. In both cases, the bit-vector encodes the occurrence of the frequent patterns as follows. Given a set of $k$ patterns, each graph $g$ is encoded as a $k$-dimensional binary vector, where a 1 is marked in the $i$-th position if the $i$-th subgraph is subgraph isomorphic to $g$ and a 0 otherwise (see Sect. 2.3.3). We have aimed at obtaining a similar number of features by selecting an appropriate frequency threshold for the mining algorithms: 4% for FOG, leading to 1376 patterns and 5% for gSpan, leading to 1292 patterns.

We use the SVM$^{light}$ implementation [Joachims, 2002]. For all methods, we use exactly the same settings, which involves applying a polynomial kernel of degree 2, using a 10-fold stratified cross-validation. For each fold, the regularisation parameter is tuned by holding out a development set from each training fold of the cross-validation. Finally, we combine the predictions from each test fold, rank

[3] `http://openbabel.sourceforge.net`

all the predictions and compute again the AUROC.

**Statistical significance** We compute the statistical significance of the different methods in two ways. To estimate the significance of the AUROC comparison between two classifiers, we use the (two-sided) Wilcoxon signed rank test [Wilcoxon, 1945], which is a non-parametric alternative to the paired Student's t-test that does not make any assumption about the distribution of the measurements. In the results, we report the $p$-value of the test. To compute statistical comparisons of multiple classifiers over multiple datasets, we use the Friedman test combined with a Nemenyi post-hoc test [Demšar, 2006]. The Friedman test is a non-parametric test for statistical comparisons of multiple classifiers. It ranks the algorithms for each dataset separately, with the best performing algorithm getting the rank of 1. In case of a tie, the average rank of the tied models is assigned. Then, a Nemenyi post-hoc test is used to analyse which of the classifier's ranks differ significantly from each other: the performance is significantly different if the corresponding average ranks differ by at most the critical difference, which depends on the significance level and the number of classifiers [Demšar, 2006].

### 6.4.3 Results

**Q1: How does the predictive performance of the proposed metric based on the $\text{MCS}_{\sqsubseteq}$ compare to the same metric based on the $\text{MCS}_{\preceq}$?**

Because the algorithm of Cao et al. [2008] has an exponential behaviour (see Sect. 5.5.3), it was not always possible to compute $\text{d}_{\preceq}$ while performing $k$-nearest-neighbour classification. A time-out of 24 hours per computation of one distance (corresponding to one MCS computation) was used. On average, the computation of $\text{d}_{\preceq}$ timed out for 0.1% of the molecules. However, since the molecules for which this occurs are large, the denominator in Eq. 6.1 will be large too, which would result in large distances to these molecules. This reduces the chance of them being selected by the $k$NN algorithm, indicating that this should not have a big impact on the classification performance.

Figure 6.2 plots the AUROC of both IBL classifiers. IBL-$d_{\sqsubseteq}$ scores better than IBL-$d_{\preceq}$ on all 60 datasets. According to the (two-sided) Wilcoxon signed rank test, this is statistically significant ($p = 1.60 \cdot 10^{-11}$). The average AUROC for IBL-$d_{\sqsubseteq}$ is 0.760, while for IBL-$d_{\preceq}$ it is 0.728.

**Conclusion** The experimental results indicate that the metric based on the $\text{MCS}_{\sqsubseteq}$ is performing better than the metric based on the $\text{MCS}_{\preceq}$. This means that using the BBP subgraph isomorphism not only leads to a metric that is efficiently computable, but that also has a significantly better predictive performance when used for the classification of molecules. The metric based on the BBP subgraph isomorphism thus seems more meaningful on a chemical level.

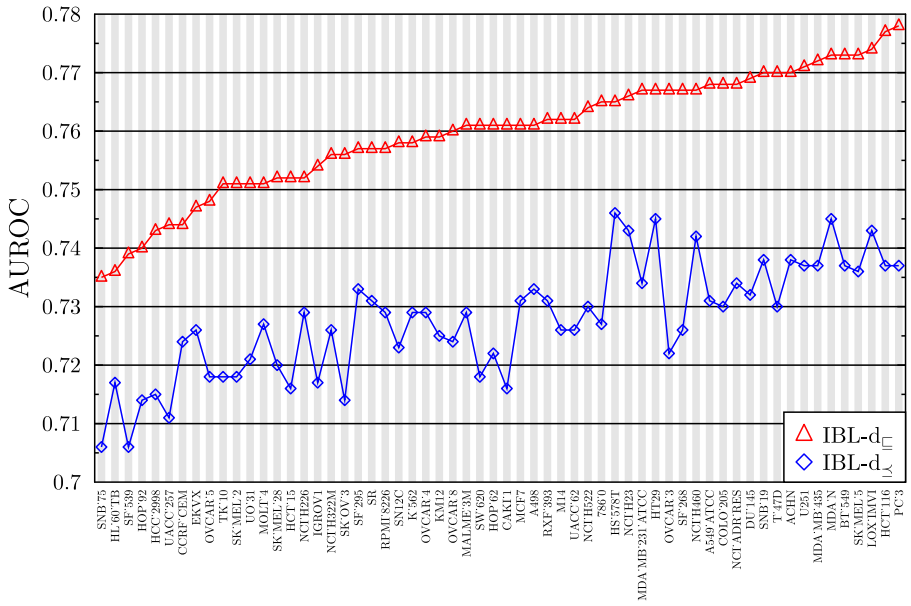


Figure 6.2: Predictive performance of the IBL-$d_{\sqsubseteq}$ and IBL-$d_{\trianglerighteq}$ classifiers on NCI60.

**Q2: What is the relationship between the best MCS-based metric and state-of-the-art metrics?**

In Fig. 6.3, we plot the AUROC of IBL-$d_{\sqsubseteq}$, IBL-FP and IBL-WDK. We find that IBL-$d_{\sqsubseteq}$ performs consistently better (60 wins out of 60) than the other two methods. According to the Friedman test (Table 6.1), which shows that the difference in average ranks is larger than the critical difference at 1%, this is statistically significant. Table 6.1 also shows the average AUROC for the different methods.

**Conclusion** IBL-$d_{\sqsubseteq}$ is performing significantly better than the current state-of-the-art metrics for molecules.

**Q3: What is the usefulness of the BBP subgraph isomorphism as matching operator in general?**

Figure 6.4 shows a similar comparison between the SVM-based classifiers on the 60 datasets. SVM-FOG scores better than SVM-gSpan on all datasets. According to the (two-sided) Wilcoxon signed rank test, this is statistically significant ($p = 1.60 \cdot 10^{-11}$). The average AUROC for SVM-FOG is 0.762, while for SVM-gSpan

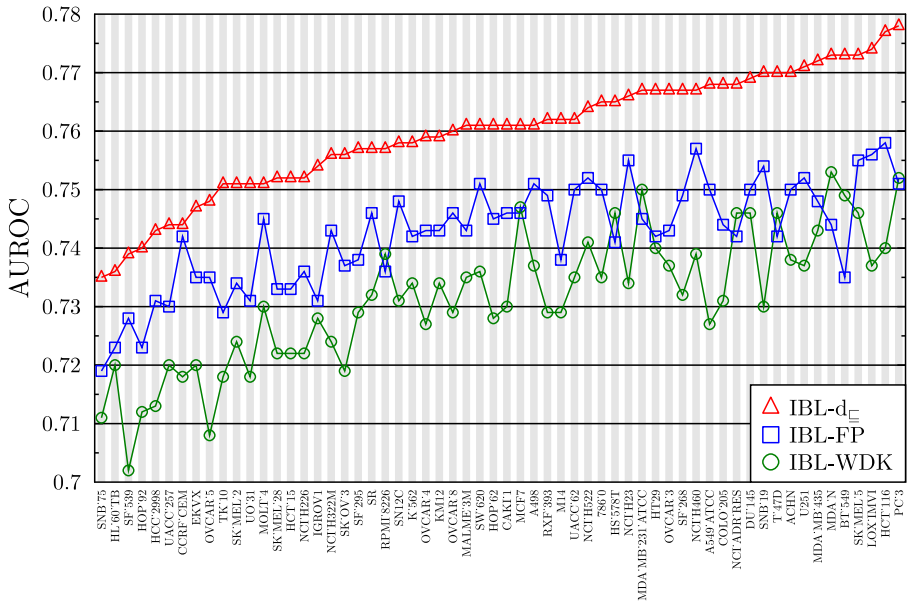


Figure 6.3: Predictive performance of the state-of-the-art IBL classifiers on NCI60.

it is 0.737.

**Conclusion** The features generated by the BBP matching operator have a larger predictive power. This can be explained by the fact that SVM-FOG uses a more constrained language, leading to less redundant patterns. This will be investigated more thoroughly in the next chapter. Additional experiments show that it is still possible to boost the performance of SVM-FOG and SVM-gSpan by lowering the support threshold (and in this way obtaining more patterns), but this does not change the above conclusion.

## 6.5 Conclusions

In this chapter, we have shown that it is possible to construct an efficiently computable metric by using the polynomial MCS algorithm of the previous chapter. We have also investigated the predictive performance of this metric and of the BBP matching operator in general.

It turns out that the BBP-based metric outperforms the metric that is based on the MCS computed under the general subgraph isomorphism. We have also shown

Table 6.1: Average scores and ranks for AUROC for the state-of-the-art IBL classifiers on NCI60.

| Method | Average AUROC | Average rank |
|---|---|---|
| IBL-$d_{\sqsubseteq}$ | 0.760 | 1 |
| IBL-FP | 0.742 | 2.15 |
| IBL-WDK | 0.731 | 2.85 |
| Critical difference for the average ranks at the 1% significance level: 0.53 | | |

that the BBP-based metric outperforms previously published metrics. Moreover, a metric based on the MCS is more intuitive than other metrics and as it uses the original graph structure. Experimental results show that the BBP matching operator obtains good performance as well when used to generate features. One reason may be that dealing differently with cycles and linear fragments makes more sense in chemical applications. Therefore, depending on the situation (e.g., the number of examples) and the user's preferences (e.g., the interpretability of the predictions), either a BBP-metric (IBL) or BBP-generated features can be good choices.

We can conclude that, at least for molecular datasets, the BBP matching operator can, next to the gain in efficiency, also improve the predictive performance of graph mining techniques and hence, it is an interesting matching operator for molecules.

The metric can also be used in other learning tasks, such as clustering. For example, when trying to interpret a dataset of molecules, it is useful for chemists to be able to cluster similar molecules in order to investigate shared properties.

Since a fraction of the molecules in the NCI database cannot be represented by outerplanar graphs, it is useful to investigate how the ideas of the BBP matching operator can be extended to also cover the non-outerplanar graphs. In the next chapter, we will provide a possible solution for this.

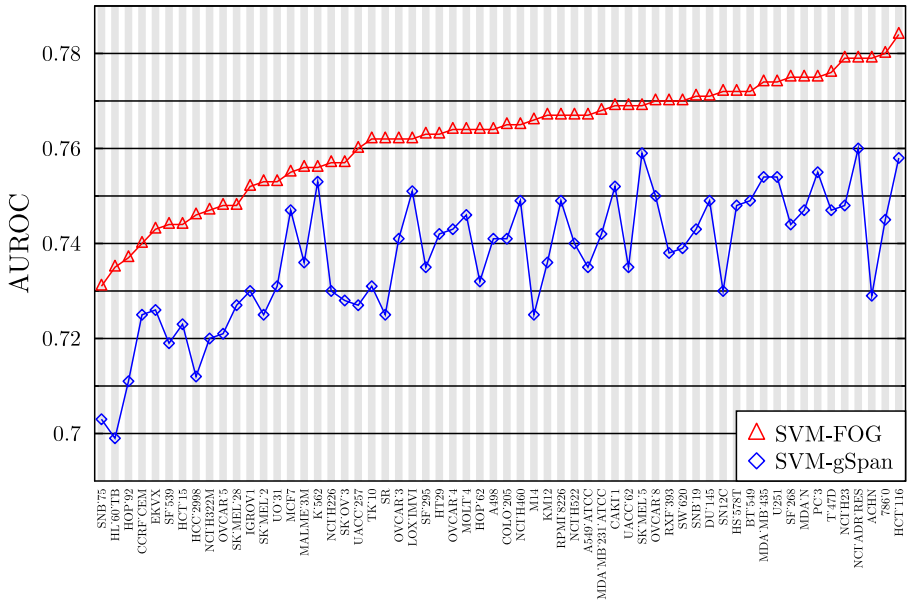


Figure 6.4: Comparison of the performance of the SVM classifiers on NCI60.

# Chapter 7

# Maximum Common Subgraph Sampling

## 7.1 Introduction

While in the previous chapter we have concentrated on using the maximum common subgraph as a similarity measure, here we turn to a different approach. We will use maximum common subgraphs as features, hereby contributing to the field of propositionalisation methods that exist for graphs and molecules in particular.

During the last decade, a lot of attention has been devoted to mining local patterns in molecular datasets, leading to the development of many graph mining systems. These systems typically employ constraints to specify the patterns of interest, such as frequency, or top-$k$ according to a correlation measure (e.g., $\chi^2$). Graph mining systems then perform a complete search through the entire graph space, enumerating all subgraphs satisfying these constraints. A number of these methods were discussed in Sect. 6.2.

Usually the resulting patterns are not used directly. Instead, they are used as features in combination with traditional machine learning algorithms. Furthermore, the quality of the generated patterns is measured through the quality of the induced classifiers or models for regression [Wale et al., 2008]. While these approaches offer strong guarantees w.r.t. completeness or optimality of the found patterns, they have a high computational cost and require post-processing to deal, for example, with redundancy issues [Bringmann et al., 2006]. In this way, local pattern mining acts as a complex, expensive and *indirect* approach to generate features for graphs.

In this chapter, we propose a simple, efficient and direct approach to generate interesting graph patterns. The idea is to compute maximum common subgraphs from randomly selected pairs of examples and to directly use them as features.

While computing maximum common subgraphs in general is an NP-hard problem, we have shown that a polynomial-time algorithm exists for outerplanar graphs in combination with the block-and-bridge-preserving (BBP) subgraph isomorphism (Chapter 5). We have argued that outerplanar graphs are a suitable class of graphs to represent molecules. Moreover, experimental results indicate that the BBP subgraph isomorphism is a sensible matching operator when classifying molecules, leading to improved predictive performance in several learning methods (Chapter 6).

The contributions of this chapter are the following:

- We introduce a framework for generating features for graphs through the extraction of maximum common subgraphs. We propose several extraction strategies and show that sampling MCS features uniformly at random is a good trade-off between efficiency and predictive performance.

- We show that the maximum common subgraphs generated in this framework yield state-of-the-art features. The advantages of this approach are 1) that it is easy to control the number of produced features (while setting the frequency in a pattern mining task yields an unpredictable number of patterns); 2) that patterns can be extracted in polynomial time and more efficiently than by frequent or correlated subgraph mining, as no search space has to be traversed; and 3) that on 60 benchmark problems from NCI, the extracted features allow for the construction of SVM classification models that achieve significantly better predictive performance than those built using features returned by traditional local pattern mining and exhaustive fingerprint generation methods.

The contents of this chapter are the result of joint work with Fabrizio Costa, Jan Ramon and Luc De Raedt and have been published in [Schietgat et al., 2010].

The chapter is organised as follows. We start by explaining our method to sample maximum common subgraphs in Sect. 7.2. Section 7.3 presents an experimental evaluation, showing multiple variants of our method and comparing them to the state-of-the-art. In Sect. 7.4, we provide a discussion about our method and finally, in Sect. 7.5, we conclude.

## 7.2 Using maximum common subgraphs as features

In this section, we describe how we use the MCS algorithm to generate features for graphs. The idea is to select pairs of graphs from the dataset, and then compute one of their MCSs. Before we discuss the method, we give a problem description.

### 7.2.1 Problem description

We define the task of generating features in graphs as follows. Consider a dataset of graphs $\mathcal{D}$, where each graph has been labeled positive or negative w.r.t. a particular classification task:

$$\mathcal{D} = \{(g_i, C_i) \mid C_i \in \{+, -\}\}.$$

Given such a dataset $\mathcal{D}$, a set of possible constraints $c$ and a number $k$ (with $0 < k < \infty$), the task is then to find a set of $k$ subgraphs satisfying the constraints $c$ that are used as features for $\mathcal{D}$.

### 7.2.2 Method

First, we introduce some additional notations. We denote with $\mathcal{D}_+$ the subset of graphs belonging to the positive class, that is,

$$\mathcal{D}_+ = \{(g_i, C_i) \in \mathcal{D} \mid C_i = +\}.$$

In the same way, we define $\mathcal{D}_-$. Note that $\mathcal{D}_+$ and $\mathcal{D}_-$ form disjoint partitions of $\mathcal{D}$, that is, $\mathcal{D} = \mathcal{D}_- \cup \mathcal{D}_+$. Then, let $\mathcal{D}^*$ be the subset of outerplanar graphs of $\mathcal{D}$, that is,

$$\mathcal{D}^* = \{g \in \mathcal{D} \mid g \text{ is outerplanar}\}.$$

Now we are ready to introduce the notation for extracting MCS features. We define

$$\mathcal{F}(X, Y) = \{p \mid p = \text{MCS}(x, y),\ x \in X,\ y \in Y\}$$

where $\text{MCS}(x, y)$ returns *an* $\text{MCS}_{\sqsubseteq}$ of $x$ and $y$ and where $X$ and $Y$ are arbitrarily defined sets of graphs. Observe that per pair of graphs, we compute only one MCS, as this guarantees a polynomial time complexity. We can obtain different subsets of $\mathcal{F}(X, Y)$ by: 1) varying the selection strategy, that is, the way we choose $X$ and $Y$, possibly using a sampling method, and 2) adding additional constraints on the found subgraphs $p$. In particular, the choices that we consider are:

1. **Selection strategies** on $X$ and $Y$
   - $X = \mathcal{D}^*$ and $Y = \mathcal{D}^*$, that is, we compute all MCSs from all pairs of outerplanar graphs in our set;
   - $X = \mathcal{D}^*_+$ ($\mathcal{D}^*_-$) and $Y = \mathcal{D}^*_+$ ($\mathcal{D}^*_-$), that is, we consider only subgraphs common between graphs belonging to the same class (either positive or negative) in order to capture features that are more discriminative for the given target concept;
   - a sampling approach that selects couples $(x, y)$ from $X \times Y$ uniformly at random. This allows one to trade off predictive performance and efficiency as a reduced set of $k$ features can be generated more quickly. We denote a reduced set of $k$ features as $\mathcal{F}^k$.

2. **Additional constraints c** on the retrieved subgraphs $p$

   - $freq(p, \mathcal{D}) \geq f$, that is, $p$ is a subgraph of more than $f$ graphs in the dataset $\mathcal{D}$;
   - $\chi^2(p, \mathcal{D}) > t$, that is, $p$ is a subgraph occurring in more graphs from the positive (negative) class than from the negative (positive) class, where the exact threshold is derived by the $\chi^2$ score (often used to compute the significance of patterns [Bringmann et al., 2006]);
   - $size(p) \leq s$, that is, $p$ has to have a size below the threshold $s$.

   When a constraint **c** is imposed on a set of features $\mathcal{F}$ we denote the resulting set as $\mathbf{c}(\mathcal{F})$. For instance, $freq(\mathcal{F}, \mathcal{D}) \geq f$ represents the set of patterns in $\mathcal{F}$ with a frequency higher than $f$. Furthermore, we use the notation $\arg\max_k \phi(\mathcal{F}, \mathcal{D})$ to denote the top $k$ features from $\mathcal{F}$, that is, the $k$ features from $\mathcal{F}$ that score best with regard to a scoring function $\phi(\cdot)$.

### 7.2.3 Computational complexity

**Time complexity** In order to gain an understanding of the time complexity of the proposed approach, we identify and discuss four key processes: 1) the computation of the set of MCSs between two graphs $x$ and $y$; 2) the selection strategy, which determines the set of graphs from which to sample the pairs $(x, y)$; 3) the elimination of multiple occurrences of the same subgraph; and 4) the embedding of the extracted subgraphs in the graph dataset:

1. **MCS computation** While computing the MCS set under the general subgraph isomorphism is NP-hard, determining a single (random) MCS under the BBP subgraph isomorphism between two outerplanar graphs can be achieved in polynomial time using the algorithm discussed in Sect. 5.3.

2. **Selection strategy** While the extraction of the MCS set from all pairs of graphs in $\mathcal{D}^*$ invokes the MCS computation a number of times quadratic in the size of the set of examples, one can hope to achieve a good compromise by either a) randomly selecting a smaller subset of graphs in $X, Y$ and invoking $\text{MCS}(x, y)$ from all possible pairs, or b) directly selecting a smaller number of random graph pairs $(x, y)$. This latter procedure raises an interesting question as to how the number of (distinct) subgraphs $k$ and the number of graph pairs $n$ relate (as the same MCS can be extracted from different graphs). This is investigated experimentally in Sect. 7.3.4, where we show that the relationship between $k$ and $n$ is just linear.

3. **Eliminating multiple MCS occurrences** To avoid multiple occurrences, we have to check for each new pattern whether it is isomorphic to an already found MCS. Because of the BBP subgraph isomorphism and the fact that

all MCSs are outerplanar, this can also be realised in polynomial time and has to be repeated $k^2$ times with $k$ the cardinality of the MCS set.

4. **Feature embedding** Once the set of $k$ MCSs has been identified, the translation of these subgraphs into features is accomplished by doing a subgraph isomorphism test between each of the $k$ elements in the MCS set and each of the $m$ elements in $\mathcal{D}^*$. Using the BBP decomposition notion, this can be done in polynomial time for each of the $k \cdot m$ pairs. To compute the embeddings for the non-outerplanar graphs, that is, for every $g \in \mathcal{D} \setminus \mathcal{D}^*$, the (NP-complete) general subgraph isomorphism test can be used.

Hence, the overall complexity is polynomial in the size of the individual graphs, in the size of the graph set and in the size of the desired feature set (which is bounded by the square of the size of the graph set).

Notice that, in contrast to traditional local pattern mining approaches, the proposed technique does not require one to perform expensive embedding operations while searching for features, but only once the features have been generated, that is, while local pattern mining techniques need to compute frequencies or correlation measures and therefore need to perform embedding computations *during* the search phase, our approach computes the embedding only *after* the whole set has been extracted.

**Space complexity** In order to gain an understanding of the space complexity of the proposed approach, we identify and discuss two key processes: 1) the space requirements for the extraction of an MCS of two graphs and 2) the space requirements when processing the entire set of examples.

In the first case, the MCS algorithm requires to store a number of relevant subgraphs bounded by $O(m^2)$ with $m$ being the number of vertices in the largest block. We note that in practice this does not imply a severe memory requirement for applications in chemoinformatics.

In the second case, the key process is the check for multiple MCS occurrences. For this we need to keep track of all unique MCS patterns found. In Sect. 7.3.4, we empirically show that the number of unique MCS patterns grows linearly w.r.t the number of examples and not quadratically as one would intuitively expect. This property allows us to conclude that the memory requirements are in practice linear w.r.t. the dataset size.

A C++ implementation of the presented method can be downloaded at `http://www.cs.kuleuven.be/~dtai/PMCSFG`.

## 7.3 Experiments

In this section, we perform an experimental analysis to measure the quality of the patterns under the various parametric choices and the computational time needed

to generate them. In particular, we want to answer the following questions:

**Q1** What is the predictive quality of MCS features obtained under different selection strategies?

**Q2** What are the effects of applying different constraints on the obtained MCS features?

**Q3** How does the quality of the feature set vary w.r.t. the number of sampled MCS features?

**Q4** How many pairs of molecules need to be sampled in order to obtain $k$ unique MCS features?

**Q5** How does MCS feature construction compare with state-of-the-art feature construction methods?

**Q6** What are the runtimes of the MCS feature generation methods and how do they compare with state-of-the-art feature construction methods?

The properties and the quality of the extracted subgraphs are evaluated by using them as features in predictive tasks for several problems from chemoinformatics. We compare the results against several related state-of-the-art methods and provide a discussion.

### 7.3.1 Datasets

We will mosty use the NCI60 datasets for our experiments, but for some comparisons, we will also show results on additional datasets.

**NCI60** The 60 datasets from NCI, providing the ability to inhibit the growth of human tumor cell lines for thousands of molecules, were discussed in Sect. 6.4.1.

**HIV** The HIV dataset contains a large number of molecules for which it was checked if they provided protection against HIV-1. The October 1999 release of the database[1] contains the structures of 42687 molecules. Each of the compounds was tested twice: 422 were confirmed to be active (CA), 1081 are moderately active (CM), and 41184 are inactive (CI). Sometimes, a subset of 41768 molecules introduced by Kramer et al. [2001] is used to benchmark machine learning algorithms. The full set has also been used before, e.g. by Ceroni et al. [2007]. There are three classification tasks commonly considered for this dataset: distinguishing between CA and CM, between (CA $\cup$ CM) and CI, and between CA and CI.

[1] The data can be downloaded from `http://dtp.nci.nih.gov/docs/aids/aids_data.html`.

Table 7.1: Percentage of non-outerplanar examples in the NCI, HIV, PTC and Bursi datasets. For HIV, percentages are reported for the whole dataset as well as the positive subset.

| Dataset | Number of examples | Number of non-outerplanar examples |
|---|---|---|
| NCI | 3910 | 379 (9.70%) |
| HIV (all) | 42687 | 3673 (8.61%) |
| HIV (pos) | 1503 | 88 (5.85%) |
| PTC | 417 | 15 (3.60%) |
| Bursi | 4337 | 361 (8.32%) |

**PTC** The 2000-2001 Predictive Toxicology Challenge (PTC) [Toivonen et al., 2003] was devised to stimulate the development of machine learning techniques for predictive toxicology models. The data originates from the US National Toxicology Program (NTP). The training and test sets have a different class distribution and a different prevailing mode of action [Benigni and Giuliani, 2003], therefore we only use the (corrected) training set, which contains 417 molecules. The aim is to predict the carcinogenicity of the molecules in different rodents, in particular male mice (MM), female mice (FM), male rats (MR), and female rats (FR).

**Bursi** Kazius et al. [2005] have constructed a dataset of 4337 molecular structures with corresponding Ames data.[2] Ames is a short-term in vitro assay designed to detect genetic damage caused by chemicals and has become the standard test to determine mutagenicity. The distribution is 2401 mutagens and 1936 nonmutagens.

**Outerplanarity** For all datasets, we report the percentage of non-outerplanar examples in Table 7.1. For HIV, we are particularly interested in the (small) positive subset, as we will investigate the extraction of features on positive examples alone.

### 7.3.2 State-of-the-art methods

We compare our method against five state-of-the-art methods. Four of them construct features for graphs, while the fifth method is a graph kernel used for the classification of molecules. These methods were briefly described in Sect. 6.2; here we discuss them in more detail while using the notation introduced in Sect. 7.2.2.

First, we consider a correlated graph miner [Bringmann et al., 2006], which traverses a search space in order to find the top-$k$ correlated graph patterns. Here,

[2] Available at `http://www.cheminformatics.org/datasets/bursi/`.

each pattern receives a $\chi^2$-correlation score w.r.t. the class value. It is known that correlated subgraph miners outperform frequent subgraph miners, which mine patterns under the frequency constraint, in terms of predictive performance [Bringmann et al., 2006]. We call this method C-GP. In our notation, it corresponds to $\arg\max_k \chi^2(\mathcal{G}, \mathcal{D})$, where $\mathcal{G}$ denotes the set of all possible graphs.

Second, we consider the method proposed by Wale et al. [2008], which generates all possible graph patterns that occur at least once in the dataset. The subgraph size is upper-bounded by a user defined parameter $s$. Wale et al. [2008] have shown that their method outperforms earlier methods such as graph kernels and fingerprints. We call this method A-GP. In our notation, it corresponds to $size(freq(\mathcal{G}, \mathcal{D}) \geq 1) \leq s$.

Third, we consider a method that computes the FP2 fingerprints (generated using OpenBabel v2.1.1[3]). This is an exhaustive method that generates all possible paths (linear sequences) up to length $s = 7$. Moreover, it makes use of basic chemical knowledge to label paths linked to a cycle and to discard uninformative paths. Because even for small $s$ (say 7 or 8) this rapidly leads to vast numbers of features, they are typically compressed into a fingerprint using a kind of hashing of the occurrences of the paths onto a fixed-length vector [Willett, 2006]. In this step, information is lost as it becomes impossible to find out which patterns are involved in the fingerprint. Despite this drawback, fingerprints are considered state-of-the-art among chemists [Willett, 2006]. We call this method FP2. In our notation, it corresponds to $size(freq(\mathcal{S}, \mathcal{D}) \geq 1) \leq s$, where $\mathcal{S}$ denotes the set of sequences.

Fourth, we include a method that generates trees [Maunz et al., 2009]. Instead of resorting to a top-$k$ approach, the size of the resulting feature set is limited by, apart from employing the usual frequency and correlation constraints, defining so-called backbone refinement equivalence classes and only allowing patterns from different classes. We call this method C-TP. In our notation, it corresponds to $\chi^2(freq(\mathcal{T}^*, \mathcal{D}) \geq f) \geq p$, where $\mathcal{T}^*$ denotes the set of trees from different backbone refinement equivalence classes.

Fifth, we consider the weighted decomposition kernel (WDK) introduced by Ceroni et al. [2007]. In the WDK, the neighborhood of a given radius is first associated to each vertex in a graph. The WDK is then computed as the product of an exact matching kernel over the vertex label with a kernel over the neighborhood edge multiset.

### 7.3.3 Method

We consider a variety of parametric choices as detailed in Sect. 7.2.2. Table 7.2 gives an overview of the variants that will be investigated as well as an overview of the state-of-the-art methods we will compare to.

[3] `http://openbabel.sourceforge.net`

Table 7.2: Overview of the different parametric choices for $\mathcal{F}$ and the state-of-the-art methods. OP is used as abbreviation for outerplanar.

| Abbreviation | Method | Language | Hashing |
|---|---|---|---|
| *MCS extraction strategies* | | | |
| A-MCS | $\mathcal{F}(\mathcal{D}^*, \mathcal{D}^*)$ | OP graphs | no |
| P-MCS | $\mathcal{F}(\mathcal{D}^*_+, \mathcal{D}^*_+)$ | OP graphs | no |
| N-MCS | $\mathcal{F}(\mathcal{D}^*_-, \mathcal{D}^*_-)$ | OP graphs | no |
| R-MCS | $\mathcal{F}^k(\mathcal{D}^*, \mathcal{D}^*)$ | OP graphs | no |
| F-MCS | $\arg\max_k freq(\mathcal{F}(\mathcal{D}^*, \mathcal{D}^*), \mathcal{D})$ | OP graphs | no |
| C-MCS | $\arg\max_k \chi^2(\mathcal{F}(\mathcal{D}^*, \mathcal{D}^*), \mathcal{D})$ | OP graphs | no |
| *State-of-the-art methods* | | | |
| C-GP | $\arg\max_k \chi^2(\mathcal{G}, \mathcal{D})$ | graphs | no |
| C-TP | $\chi^2(freq(\mathcal{T}^*, \mathcal{D}) \geq f) \geq p$ | trees | no |
| A-GP | $size(freq(\mathcal{G}, \mathcal{D}) \geq 1) \leq s$ | graphs | no |
| FP2 | $size(freq(\mathcal{S}, \mathcal{D}) \geq 1) \leq s$ | sequences | yes |

A-MCS corresponds to extracting MCSs from all outerplanar examples, while P-MCS and N-MCS only extract MCSs from positive or negative examples alone, respectively. R-MCS corresponds to extracting MCSs from randomly sampled pairs of outerplanar graphs, while F-MCS and C-MCS first extract all MCSs and then keep the top $k$ ones w.r.t. frequency and $\chi^2$, respectively.

**Parameter settings for MCS extraction strategies** For R-MCS, F-MCS and C-MCS, we chose $k = 1000$. Since R-MCS is a non-deterministic method, it was always run 10 times and boxplots are reported. For F-MCS and C-MCS, we also chose $k = 1000$. For all MCS methods, we discard subgraphs that only have a single vertex, as was done by Bringmann et al. [2006].

**Parameter settings for state-of-the-art methods** For C-GP, we chose $k = 1000$ and mined the top 1000 most correlated patterns in the training data. For A-GP, we consider all subgraphs from length 1 to 7, that is, we chose $s \leq 7$, as recommended by Wale et al. [2008]. FP2 also uses $s \leq 7$ and requires one additionally to specify the number of bits for the pattern encoding vector. A common choice for this number is 10, and since 1024 (the length of the vector) is closest to the value of $k$, we adopt the same value of 10. For R-TP, we used a frequency threshold of 2% and a correlation threshold of 95%, as suggested by the author. For WDK, we used a radius of 4 and combined the WDK kernel with a polynomial kernel of degree 5, motivated by the results of Menchetti et al. [2005].

**Evaluation** Since we want to investigate the predictive quality of different feature generation methods, we vary only the feature generation step and resort to the same classification procedure for all the feature generation methods.

Given a graph dataset $\mathcal{D}$, we first generate features only from the training set. Then, we propositionalise each example in $\mathcal{D}$ to a one-bit vector encoding representation: given a feature set of size $k$, each graph $g \in \mathcal{D}$ is encoded as a $k$-dimensional binary vector, where a 1 is marked in the $i$-th position if the $i$-th subgraph is subgraph isomorphic to $g$. This propositionalisation technique was discussed in Sect. 2.3.3. The general subgraph isomorphism is used for this matching for all methods.

As classification model we use SVMs in combination with the Tanimoto-kernel [Swamidass et al., 2005]:

$$\mathcal{K}_T(x,y) = \frac{\sum_{i=1}^{N}(x_i = 1 \wedge y_i = 1)}{\sum_{i=1}^{N}(x_i = 1 \vee y_i = 1) - \sum_{i=1}^{N}(x_i = 1 \wedge y_i = 1)}$$

In words, this kernel computes a similarity between vector $x$ and vector $y$ by counting the number of common patterns (i.e. the set-intersection) between the two molecules as a fraction of the total number of patterns that occur in both molecules (i.e. the set-union), which is similar to the Jaccard coefficient. The Tanimoto-kernel is considered state-of-the-art for the classification of small molecules [Willett, 2006]. As implementation we used SVM$^{light}$ [Joachims, 2002].

To evaluate the classification models, we use the area under the ROC-curve (AUROC) score [Provost and Fawcett, 1998].[4] For all experiments, a stratified 10-fold cross-validation is used. The regularisation parameter of the SVM is tuned out of 10 values running an internal 5-fold cross-validation over the training data.

We compute the statistical significance of the different methods from their wins and losses. In particular, we use the Friedman test combined with a Nemenyi post-hoc test to compute significance [Demšar, 2006]. The Friedman test was explained in Sect. 6.4.2.

### 7.3.4 Results

We organise the experimental results as answers to the set of six questions.

**Q1: What is the predictive quality of MCS features obtained under different selection strategies?**

Figure 7.1 shows the predictive performance for A-MCS, P-MCS and N-MCS on the 60 NCI datasets. On average, A-MCS resulted in approximately 7800 pat-

[4]Since Hand [2009] shows that AUROC fails to take into account the relative costs of misclassifications of different classifiers, we also computed the H-measure, which Hand [2009] proposes as an alternative. However, since this does not lead to different conclusions for all experiments, we do not report the results w.r.t. the H-measure.

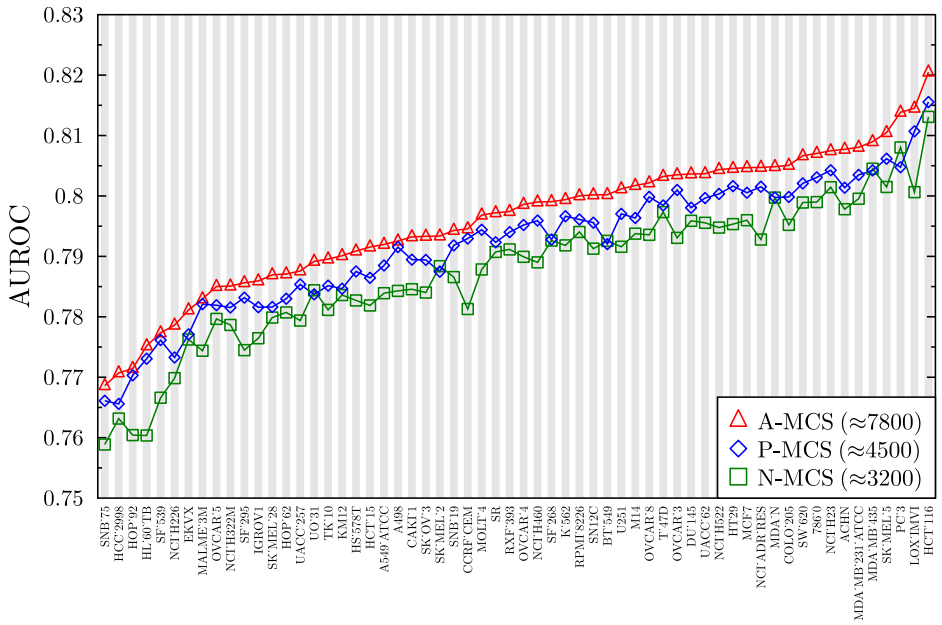


Figure 7.1: Predictive performance of different selection strategies on NCI60.

terns, P-MCS in 4500 patterns and N-MCS in 3200 patterns. According to the Friedman test, for which the average ranks and critical difference are shown in Table 7.3, A-MCS is significantly outperforming P-MCS and N-MCS. P-MCS in turn outperforms N-MCS.

However, the average AUROC scores over the 60 datasets (also shown in Table 7.3), are not very different. This is an interesting result since, when there are reasons to believe that the negative class is more complex to model or when the available dataset has a larger number of negative examples (conditions that often occur in molecular activity prediction tasks), one can resort to sampling from positive examples alone without losing much predictive performance in practice.

**Conclusion** Extracting MCS features of positive (or even negative) examples results in significantly worse predictive performance, but still obtain good results in practice. Further investigation suggests that the small decrease in performance of P-MCS and N-MCS is likely to be caused by the reduced number of features (see also Q3).

Table 7.3: Average scores and ranks for AUROC over the 60 NCI datasets when comparing different selection strategies.

| | AUROC | |
|---|---|---|
| Method | Average | Average rank |
| A-MCS | 0.796 | 1 |
| P-MCS | 0.792 | 2.08 |
| N-MCS | 0.788 | 2.92 |
| Critical difference for the average ranks at the 1% significance level: 0.53 | | |

Table 7.4: AUROC for the three classification tasks of the HIV dataset.

| Method | CA vs. CM | CACM vs. CI | CA vs. CI |
|---|---|---|---|
| P-MCS | 0.826 | 0.818 | 0.930 |
| WDK | 0.831 | 0.829 | 0.940 |

**Case study** To test the performance of P-MCS in practice, we have run it on the HIV datasets, which are highly skewed: the number of inactive examples (41184) highly exceeds the number of active (422) and moderately active (1081) examples. Table 7.4 shows a comparison of P-MCS with WDK. By extracting MCSs only over the 1503 positive examples, the computation time is greatly reduced, while the predictive performance is close to state-of-the-art results reported on this dataset.

**Q2: What are the effects of applying different constraints on the obtained MCS features?**

To answer this question, we will compare a random sample of 1000 MCS features (R-MCS), that is, applying no constraint at all, to the 1000 most frequent MCS features (F-MCS) and the 1000 most correlated MCS features (R-MCS). The predictive performance of R-MCS, F-MCS and C-MCS is shown in Fig. 7.2. Note that, because for R-MCS the results are averaged over 10 runs, boxplots are shown. These show that, despite the non-deterministic nature of the procedure, R-MCS achieves quite stable results.

The Friedman test (results shown in Table 7.5) shows a clear advantage for R-MCS over F-MCS and C-MCS. However, again the average AUROC scores indicate that there is no large difference in practice (Table 7.5).

**Conclusion** The results show that extracting MCS features from randomly sampled pairs of examples results in a significantly better predictive performance than

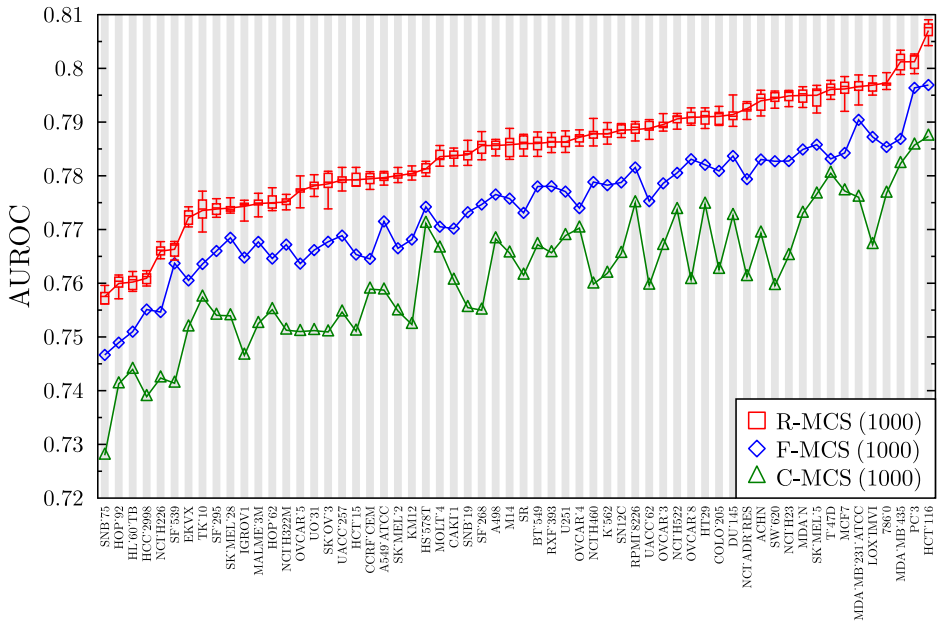


Figure 7.2: Predictive performance when applying different constraints on NCI60.

applying a frequency or correlation constraint on the MCS features. This is a surprising result, since randomly sampling 1000 MCSs is less computationally expensive (see Q4) than mining *all* MCSs and then post-processing those under some constraint to obtain the 1000 best features. A possible reason for this is that constraints tend to decrease the diversity of the set, i.e. features that are highly frequent or correlated with the target could be highly inter-correlated and hence redundant and ultimately uninformative. We further investigate this issue in the following paragraph.

**Redundancy issues** In order to gain a deeper understanding on the quality and the differences between the various feature sets, we define some indicators reported in Table 7.6. First, we define *uniqueness* as the percentage of examples with a bit-vector encoding that is unique, i.e. different from that of all the other examples in the dataset. It is evident that examples having the same encoding cannot be further discriminated by any classification method. Hence, a high uniqueness is a desirable property.

Second, we report the generalisation of the mutual information measure, i.e. the *total correlation* [Watanabe, 1960] to express the amount of redundancy existing

Table 7.5: Average scores and ranks for AUROC over the 60 NCI datasets when applying different constraints.

| | AUROC | |
|---|---|---|
| Method | Average | Average rank |
| R-MCS | 0.784 | 1 |
| F-MCS | 0.774 | 2 |
| C-MCS | 0.761 | 3 |
| Critical difference for the average ranks at the 1% significance level: 0.53 | | |

among the set of features considered as random variables. The total correlation (TC) is defined as:

$$TC(X_1, X_2, \ldots, X_k) = \sum_{i}^{k} H(X_i) - H(X_1, X2, \ldots, X_k)$$

for the set of $k$ features, where $H(\cdot)$ is the (joint) information entropy. It represents the amount of information shared among the variables in the set. The sum $\sum_i^n H(X_i)$ represents the amount of information (in bits) that the features would possess if they were totally independent of one another. $H(X_1, X_2, \ldots, X_n)$ is the actual amount of information that the feature set contains. The difference between these terms therefore represents the absolute redundancy present in the given set of features, that is, the TC tells us how related a group of features are. A near-zero TC indicates that the features are essentially statistically independent; they are completely unrelated, in the sense that knowing the value of one feature does not provide any clue as to the values of the other features. A maximum value for TC is achieved when one of the features is completely redundant with respect to all of the other features.

We argue that a good set of features should have 1) a high uniqueness (so to be injective and not commit to some predefined, target independent equivalence notion between examples) and 2) a low total correlation (i.e., a low amount of information shared among the features).

All results shown in Table 7.6 have been averaged over the 60 datasets and only test set examples, that were not used for the generation of the patterns, were considered. Since we do not have access to the actual patterns that were generated by FP2, this method is not included in the table. Because the total correlation of different features sets only has a valid interpretation when dealing with the same amount of features, we do not report it for the $10^5$ features of A-GP or for the 7800 features of A-MCS. According to these indicators, R-MCS selects a better set of features than the other strategies. Moreover, we have also observed that R-MCS returns features with a high frequency (occurring on average in 1/3 of the

Table 7.6: Redundancy evaluation of the different feature construction methods (averaged over the 60 NCI datasets) that yield 1000 features.

| Method | Uniqueness | Total Correlation |
|---|---|---|
| A-MCS | 99.19± 0.18 | N/D |
| R-MCS | 98.52± 0.26 | 103.55 ± 1.26 |
| F-MCS | 97.00± 0.45 | 148.57 ± 2.74 |
| C-MCS | 91.38± 2.11 | 139.32 ± 2.59 |
| A-GP | 99.36± 0.20 | N/D |
| C-GP | 53.92± 5.74 | 212.44 ± 16.65 |

Table 7.7: Distribution of the number of edges of three feature generation methods for a representative dataset (786_0). O.$k$ represents the $n \cdot k$ order statistic of the distribution.

| Method | Average | Min. | O.05 | O.25 | O.5 | O.75 | O.95 | Max. |
|---|---|---|---|---|---|---|---|---|
| A-MCS | 13.22 ± 7.56 | 1 | 5 | 9 | 12 | 16 | 27 | 98 |
| A-GP | 6.46 ± 0.86 | 1 | 5 | 6 | 7 | 7 | 7 | 7 |
| C-GP | 9.86 ± 3.80 | 1 | 5 | 8 | 10 | 12 | 16 | 19 |

test set), showing that with high probability, computing MCSs between randomly chosen pairs of graphs leads to features that are also frequent.

We finally report in Table 7.7 some order statistics on the edge set size distribution of the subgraphs retrieved with A-MCS, A-GP and R-GP. A-MCS shows a clear preference in selecting significantly larger (and perhaps more interesting) subgraphs.

**Conclusion** The features generated by A-MCS and R-MCS seem to have a larger uniqueness and are less redundant, which likely contributes to their superior predictive performance.

**Q3: How does the quality of the feature set vary w.r.t. the number of sampled MCS features?**

We measure the quality of the feature set as the predictive performance over 5 randomly selected datasets as we increase $k$, the number of randomly sampled MCSs, from 100 to 6400 (each result has been averaged over 10 runs). For this experiment we do not tune the regularisation parameter of the SVM, but take a fixed value equal to 1 (this was the best-performing parameter value in the

Table 7.8: Predictive performance (AUROC) on 5 NCI datasets with an increasing number of randomly sampled MCSs.

| | Number of MCS features | | | | | | |
|---|---|---|---|---|---|---|---|
| Dataset | 100 | 200 | 400 | 800 | 1600 | 3200 | 6400 |
| SNB_19 | 74.3 | 76.0 | 77.2 | 78.0 | 78.8 | 79.2 | 79.4 |
| M14 | 74.3 | 76.0 | 77.6 | 78.7 | 79.5 | 80.0 | 80.2 |
| NCI_H522 | 74.9 | 76.3 | 77.6 | 78.7 | 79.5 | 80.1 | 80.3 |
| 786_0 | 75.1 | 77.1 | 78.3 | 79.4 | 80.1 | 80.4 | 80.7 |
| HCT_116 | 76.2 | 78.0 | 79.4 | 80.4 | 81.2 | 81.7 | 82.0 |

previous experiments). Table 7.8 shows an AUROC improvement of ≈5% when increasing the number of patterns from 100 to 6400 with a saturation level around 3200 patterns.

**Conclusion** For R-MCS, our intuition that using more features boosts predictive performance is correct. Note that with very few patterns, it is already possible to obtain a reasonable predictive performance.

### Q4: How many pairs of molecules need to be sampled in order to obtain $k$ unique MCSs?

We experimentally determine the functional link between the number of examples and the number of unique MCSs by considering two strategies. In the first strategy, we take subsets of $n$ examples and consider all $n(n-1)/2$ possible pairs of which we compute an MCS (S1). In the second strategy, we consider a random sample of $m$ pairs from the set of all examples (S2). This corresponds to R-MCS.

The results of the two strategies are reported in Fig. 7.3. We observe that S2 needs to consider less pairs to obtain the same amount of unique MCSs, confirming the intuition that the repeated use of the same molecule in different pairs yields a smaller number of unique MCSs. Specifically, we found that in order to obtain 1000 different MCSs we need 45,000 pairs of randomly sampled molecules or a random sample of 400 molecules out of which to consider all possible pairs. We have observed an almost perfect linear relationship (with coefficient 2.6) between the number of molecules and the number of different MCSs, that is, given a set of 1000 molecules, extracting the MCSs from all pairs gives 2,600 unique MCSs.

The reason is that the number of distinct MCSs does not grow linearly, but rather as the square root of the number of pairs of examples as shown in Fig. 7.3. The explanation for this behavior is subject of current study, but it seems to

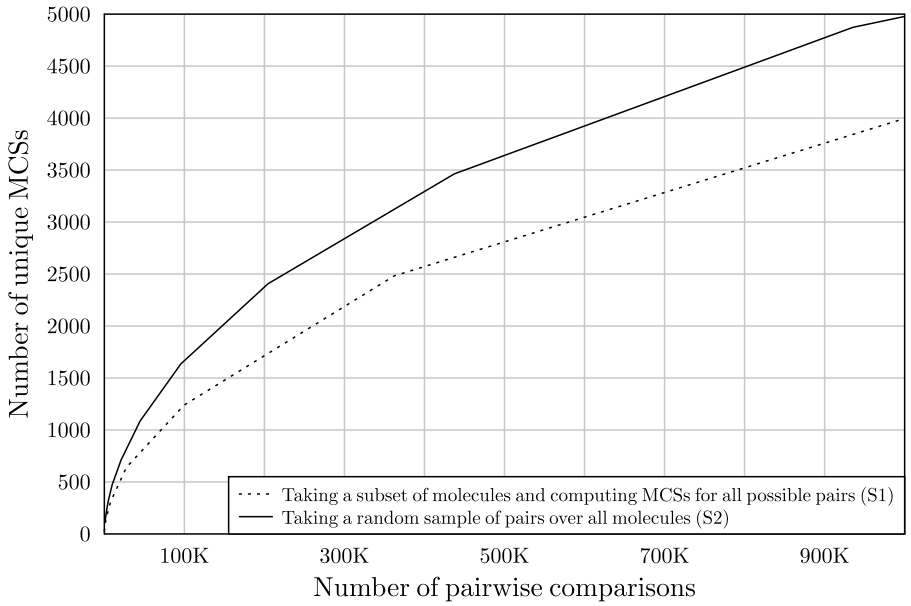


Figure 7.3: Relationship between the number of pairwise comparisons and the number of unique MCSs.

be related to the specific highly combinatorial nature of subgraphs[5] which biases shorter subgraphs to occur exponentially more frequently, which in practice, greatly reduces the number of different MCSs actually present.

**Conclusion** When considering a sampling approach, it is better to take the full set of examples into account and consider random pairs, rather than computing MCSs of all pairs on a selected subset of examples.

### Q5: How does MCS sampling compare with state-of-the-art feature generation methods?

We first compare R-MCS (results are again averaged over 10 runs) to C-TP and C-GP over the 60 datasets. Figure 7.4 shows a clear advantage for R-MCS. Also the Friedman test (Table 7.9) shows that R-MCS is performing significantly better

[5]A similar behavior is observed in the growth of the number of distinct words (which are sequences of atomic letters) in natural texts.

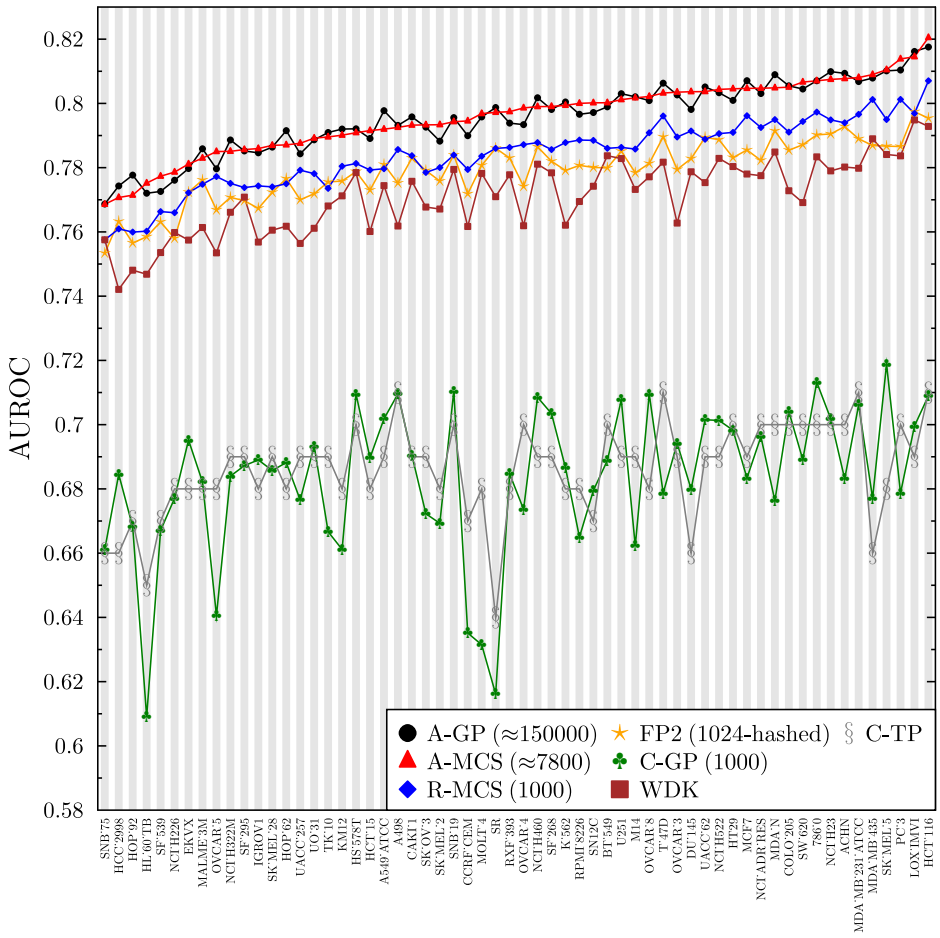


Figure 7.4: Predictive performance of state-of-the-art feature generation methods on NCI60.

than C-TP and C-GP.[6]

Next, we compare R-MCS with FP2 (Fig. 7.4). Here, the Friedman test indicates that there is no significant difference between these two methods (Table 7.9).

[6]Bringmann et al. [2006] argue that mining the top-$k$ sequences introduces less redundancy in the patterns than mining the top-$k$ graphs. We have therefore tested the latter approach which yields an average AUROC of 0.736, still significantly below that of R-MCS.

Table 7.9: Average scores and ranks for AUROC over the 60 NCI datasets for the state-of-the-art feature generation methods.

| | AUROC | |
|---|---|---|
| Method | Average | Average rank |
| A-MCS | 0.796 | 1.45 |
| A-GP | 0.796 | 1.55 |
| R-MCS | 0.784 | 3.2 |
| FP2 | 0.779 | 3.9 |
| WDK | 0.771 | 4.9 |
| C-TP | 0.686 | 6.48 |
| C-GP | 0.684 | 6.52 |
| Critical difference for the average ranks at the 1% significance level: 1.36 | | |

If we compare R-MCS to WDK, we find that R-MCS significantly outperforms WDK according to the Friedman test (Table 7.9).

Finally, we compare A-MCS with A-GP. Figure 7.4 shows that both methods are competitive in terms of predictive performance. The outcome of the Friedman test (Table 7.9) also indicate that the performance of A-MCS and A-GP are not significantly different. However, A-GP needs ≈150,000 patterns to reach this performance, while A-MCS needs only ≈7,800 patterns. Moreover, it can be argued that, because of the BBP subgraph isomorphism, the patterns of A-MCS are more easily interpretable from a chemical viewpoint.

To further investigate the quality of both sets of patterns, we have randomly selected 1000 patterns from the approach of Wale et al. [2008] (R-GP) and compared the decrease in predictive performance. For R-GP, the average AUROC was 0.679. It turns out that GP degrades much more than MCS: the decrease in average AUROC (A-GP – R-GP) is 11.7, while for our approach (A-MCS – R-MCS) it is only 1.1. This shows that MCS features are more robust and meaningful. To check the redundancy of the patterns generated by R-GP, we also computed their uniqueness (25.25 ± 2.24) and redundancy (12.28 ± 0.80). These numbers show that R-GP yields a set of nearly totally independent features. However, if we compare the predictive performance of R-MCS to the one of R-GP, it also becomes clear that achieving non-redundancy among the features is not the only prerequisite to generate a good set of features.

**Conclusion** A-MCS and R-MCS can be considered as state-of-the-art feature generation methods.

Table 7.10: AUROC of A-MCS on the classification tasks of PTC and Bursi.

| Method | PTC (MM) | PTC (FM) | PTC (MR) | PTC (FR) | Bursi |
|---|---|---|---|---|---|
| A-MCS | 0.651 | 0.628 | 0.649 | 0.653 | 0.905 |
| WDK | 0.639 | 0.608 | 0.643 | 0.596 | 0.880 |

**Results on other datasets** We also report results for PTC and Bursi. Unfortunately, WDK is the only method for which we also have results on these datasets. For these datasets, the regularisation parameter of the SVMs used for A-MCS was not tuned by an internal cross-validation, but instead a fixed value of 1 was used. This was the selected value for the vast majority of the cases in the NCI datasets.

**Q6: What are the runtimes of the MCS feature generation methods and how do they compare with state-of-the-art feature construction methods?**

First, we compare A-MCS with R-MCS. We executed both approaches on the set of 3910 distinct molecules from the 60 NCI datasets. The average number of vertices of these molecules was 23, the average number of edges 25. All experiments were run on an Intel Core2 Quad Q9550 CPU (2.8GHz).

A-MCS needed $3.7 \cdot 10^5$ seconds, while R-MCS needed 2,142 seconds. Obviously, A-MCS is a time-consuming task, partly because of the many tests for duplicate MCSs. R-MCS, however, has a good trade-off between predictive performance and efficiency: it is 175 times faster compared to A-MCS, while only a decrease of 1.1% in AUROC was measured. One argument in favour of A-MCS, however, is that it, unlike the other feature generation methods, can easily be run in parallel.

Second, we compare R-MCS with C-GP. We randomly selected 5 datasets from NCI60 for this experiment. R-MCS needed on average 2,327 seconds per dataset, while C-GP needed on average 54,322 seconds, which is 23 times slower than R-MCS.

**Conclusion** When handling large datasets and runtimes are important, R-MCS provides a good trade-off between predictive performance and efficiency. Moreover, R-MCS achieves a speedup of a factor 23 w.r.t. a typical correlated graph miner.

## 7.4 Discussion

In this section, we discuss possible drawbacks of our method and provide additional comments on the relation to other research in this context.

### 7.4.1 Drawbacks of the method

We have shown that features obtained as maximum common subgraphs from all pairs of instances in a dataset (or those obtained by sampling from a reduced set of pairs) allow for the construction of predictive models achieving state-of-the-art performance on several tasks in chemoinformatics. There are however some drawbacks and far reaching implications of the presented method that deserve to be further discussed.

First, we notice that efficiency in the proposed approach can be guaranteed only when restricting the data to outerplanar graphs. Indeed, if an instance is a non-outerplanar graph, then it is not considered in the feature generation process. This restriction is not particulary severe when (a) the proportion of non-outerplanar examples is very small (few percentages w.r.t. the entire dataset size) or (b) the number of cases where interesting features are themselves non-outerplanar is negligible. The first case is often true in many SAR applications, although there exist datasets where the number of non-outerplanar instances is relatively large (10-20% of the total size). In those cases, methods that can exploit all the available examples could in principle achieve better performances by simple virtue of a larger dataset from which to extract relevant features. We have experimentally investigated the consequences of point (b) by extracting the top-$k$ correlated subgraphs according to the graph miner of Bringmann et al. [2006]. In this case, we have verified that all subgraphs are indeed outerplanar. A possible explanation for this is that the found patterns are too small to form non-outerplanar graphs (see Table 7.7).

Second, we observe that by using the BBP subgraph isomorphism, ring structures will be either entirely selected as part of the MCS or not at all. As a consequence, ring structures and linear fragments are treated in a different way. According to the experimental results from this thesis, this bias seems to have positive effects on the quality of the retrieved patterns when dealing with applications from chemoinformatics. The effect of this bias on graphs in other types of domains needs to be empirically evaluated on a per-application basis.

Third, we acknowledge that extracting MCS features from all possible pairs of instances is a quadratic procedure which therefore does not scale well when dealing with large datasets. To tackle this issue, we have proposed a randomisation strategy that sacrifices predictive performance in order to speed up the process. Interestingly, we have experimentally shown that the performance of models built on the MCS features saturates rapidly with the number of different MCSs so that only a relatively small number of random pairs of instances is needed to achieve results comparable with the all-pairs case. Once again though, it is unclear if these findings hold in different domains.

### 7.4.2 Related work

As already pointed out in Sect. 6.2, there are various streams of research that are related to this work. In this section, we will point out some specific differences between the MCS extraction method we have proposed in this chapter and the state-of-the-art feature generation methods.

Our approach differs from the propositionalisation approaches in that it works bottom-up and also in that it computes pairwise minimally general generalisations of the examples that can be used as features, and it combines this idea with randomisation. It is straightforward to adapt our technique for use in e.g., logical learning. One only has to replace the use of the maximum common subgraph notion by a relational notion of minimally general generalisation [De Raedt, 2008]. It is interesting to note that He and Singh [2006] propose to rank the subgraphs returned by a frequent graph miner according to a notion of statistical significance[7] and show that in a chemical database, the selected features are typically subgraphs that are in fact the "largest common subgraphs in a class of medically effective compounds".

We conjecture that methods looking for patterns that satisfy given constraints are more subject to redundancy issues than randomised methods. The intuition here is that similar or correlated patterns do exhibit the same properties w.r.t. the constraints and are therefore more likely to be all selected in the top-$k$ set, hereby reducing the diversity of the set. Intuitively, the randomisation procedure decreases the chance to select two patterns that are related in any special way (e.g., being similar or correlated). At the same time, a randomisation procedure should not decrease the quality of the retrieved patterns. In the top-frequency case, sampling $k$ elements randomly from a larger set of top-frequent patterns leads to patterns with a lower frequency on average than those obtained by a direct top-$k$ frequent approach. Hence, the random sampling has a negative impact on the desired pattern quality, that is, the selected patterns are less frequent and potentially less relevant. In the MCS case, the random sampling does not alter the main property of a pattern being the maximum common subgraph between a pair of instances.

While completeness and optimality are interesting theoretical properties, these approaches are also computationally much more demanding and may be harder to tune (that is, set parameters) than the simple randomised approach we pursued. At the same time, the completeness and optimality properties are not directly related to the true task in these graph miners, which is concerned with finding good representations of the graphs or molecules for use in classification. Our work shows clearly that, at least for molecular applications, a much simpler approach without strong guarantees may well achieve better results both in terms of predictive performance and efficiency.

[7] The p-value for a subgraph is defined as the probability that the given subgraph occurs in a database of random graphs with a support higher than the observed frequency.

The favorable properties of randomisation approaches, in particular the fact that choosing random features can be better than choosing them according to specific criteria, have already been noted in other contexts e.g., for selecting features in distance construction [Sebag, 1997]. Recently, the randomisation idea has also been suggested in the area of pattern mining. Chaoji et al. [2008] have introduced a feature construction method that obtains good patterns by sampling under diversity constraints. However, the suggested method requires the user to tune and specify two parameters that control the diversity (orthogonality) and representativeness respectively.

The most common state-of-the-art approach to feature construction in the chemoinformatics literature is to generate all patterns of size up to $k$ (typically paths) that occur in at least one molecule [Wale et al., 2008; Willett, 2006], the so-called fingerprints. The differences with our approach are that our features are guaranteed to occur in at least two molecules, that they are typically also much more informative as their size is typically larger, and at the same time, the number of such features is much smaller.

## 7.5 Conclusions

We have introduced a simple, direct and effective approach to extracting patterns in graphs. It is based on the idea of computing the maximum common subgraph of randomly selected pairs of graphs. The approach is very efficient (it runs in polynomial time thanks to the restriction to outerplanar graphs), it does not require specifying any parameter (since one can simply extract all possible distinct pairwise MCSs), yields better sets of features than alternative approaches (as measured by the predictive performance of classifiers built using the returned subgraphs as features) and seems to produce a smaller and less redundant set of features than alternative techniques.

It was argued that the minimally general generalisation approach provides an interesting alternative to the fingerprints that are so popular in chemoinformatics today. The advantages are that one obtains significantly larger and hence chemically more meaningful patterns, as well as a smaller number of them.

Probably the most surprising finding of our work was that extracting MCSs randomly produces better features than traditional and computationally more demanding graph mining approaches. This in turn sheds new light on the traditional local pattern mining approach, which has dominated the field of data mining in the past 15 years. Our results indicate that for some tasks, such as finding interesting and representative features in molecular data, it may be better to employ simpler and more efficient approaches based on e.g., randomisation. Therefore, we hope that this work encourages more research in this direction.

# Conclusion Part II

In this part, we have shown that outerplanar graphs, which form a strict generalisation of trees, are a useful class of graphs for molecular datasets. Moreover, when using the BBP subgraph isomorphism on them as matching operator, efficient learning algorithms can be developed that achieve a state-of-the-art predictive performance.

First, we have introduced an algorithm that computes an MCS of two outerplanar graphs under the BBP subgraph isomorphism. We have proved it correctness and shown that, as opposed to MCS algorithms that use the general subgraph isomorphism, it runs in polynomial time, both theoretically and empirically.

Second, we have shown that it is possible to construct an efficiently computable metric by using the polynomial MCS algorithm. We have also investigated the predictive performance of this metric and of the BBP matching operator in general. We show that, at least for molecular datasets, the BBP matching operator can, next to the gain in efficiency, also improve the predictive performance of graph mining techniques. A possible explanation may be that dealing differently with cycles and linear fragments makes more sense in chemical applications. Moreover, the MCS-based metric is more intuitive than other metrics and although it uses the original graph structure, it is still efficiently computable.

Third, we have introduced a simple, direct and effective approach to extracting patterns in graphs. It is based on the idea of computing the maximum common subgraph of randomly selected pairs of graphs. The approach is very efficient, does not require specifying any parameter, yields better sets of features than alternative approaches and seems to produce a smaller and less redundant set of features than state-of-the-art feature generation methods for graphs. It turns out that the main problem of many graph pattern miners is feature redundancy. Typically, the set of patterns is limited by some criterion such as frequency or correlation, which increases redundancy in the set of selected patterns. Our feature generation method seems to suffer less from redundancy, which in our opinion has two reasons. First, the class of outerplanar graphs seems to be a good trade-off between a language that is expressive enough to represent molecules (unlike trees and sequences which are less intuitive for molecules), while the use of the BBP subgraph isomorphism

limits the set of features, and hopefully keeping only the most relevant ones. Second, by extracting maximum common subgraphs in a randomised manner, it is more difficult to end up with redundant patterns than when using frequency or correlation constraints.

# Conclusions

# Chapter 8

# Thesis Summary and Further Work

In this chapter, we summarise the most important results of this thesis and provide possible directions for further work.

## 8.1 Thesis summary

The work presented in this thesis is situated in the field of relational learning. We have focused on learning algorithms for structured data, which deal with graphs either as their input or output. The overall goal of this thesis was to improve the efficiency of such learning methods and to apply them to real-life applications from biology and chemistry.

### Part I: Structured Output Learning for the Prediction of Gene Function

In the first part, we have studied the task of hierarchical multi-label classification (HMC), where examples can belong to multiple classes and where the classes are organised in a hierarchy. A key application of HMC is gene function prediction: biologists have a set of possible functions that genes may have, and these functions are organised in hierarchies such as the Gene Ontology. It is known that a single gene may have multiple functions. The HMC task can be interpreted as a structured output learning problem, since the goal is to correctly predict a set of paths in the single graph representing the hierarchy. In order to fulfil the hierarchy constraint, which postulates that for every predicted class, all of its superclasses in the hierarchy should be predicted as well, each path should have the predicted class and the root of the hierarchy as its endpoints.

As learning technique we choose decision trees for several reasons. First, decision trees are efficiently learnable, even on large datasets. Moreover, computing predictions with the learned decision trees is very fast as well. Second, decision trees are intuitive and easy to explain to biologists who do not have a background in machine learning. Decision trees are also known to be interpretable, since it is possible to gain insights in the tree's predictions by looking at the attributes that were used in the tests of the tree. Third, decision trees are known to produce accurate predictions on several machine learning tasks.

We have proposed and compared several approaches that address the HMC task with decision trees: (1) an algorithm that learns a single tree predicting all classes at once (Clus-HMC), (2) an algorithm that learns a separate decision tree for each class (Clus-SC), and (3) an algorithm that learns and applies such single-label decision trees in a hierarchical way (Clus-HSC). As evaluation criterion we use precision-recall curves, because they are capable of dealing with the skewed class distributions that are characteristic for HMC tasks. By evaluating these approaches on datasets from functional genomics, we have learned that:

- Clus-HMC achieves **faster learning times**: although it takes longer to build one HMC tree compared to building one SC or HSC tree, Clus-HMC takes only a fraction of the time needed to build $|C|$ SC or HSC trees, with $|C|$ the number of classes in the hierarchy, obtaining reductions in learning time by a factor of 59 to 129.

- Clus-HMC results in **smaller trees**: although one HMC tree is larger than a single SC tree, the combination of the $n$ SC or HSC trees is a factor 300 to 1000 times larger than a single HMC tree. This benefits the interpretability of HMC trees, while HMC trees also guarantee that attribute tests are selected which are relevant for all classes at once.

- Perhaps the most surprising result is that Clus-HMC obtains a **higher predictive performance** on the function prediction tasks. Experimental analysis has shown that Clus-SC and Clus-HSC are more susceptible to overfitting. In hindsight, this makes sense, since is it much harder for a model to overfit for $n$ classes at once, rather than for just one.

Next, we have introduced Clus-HMC-Ens, the ensemble version of Clus-HMC. We have shown that the boost in predictive performance that is obtainable by ensembles carries over to the HMC context. The increased predictive performance comes at a cost, however. By constructing an ensemble of trees, the learning time and size of the resulting model increases. Moreover, it is more difficult for a domain expert to interpret the resulting model. The trade-off between predictive performance and efficiency can be controlled by the parameter $k$, representing the number of trees in the ensemble. More importantly, we have shown that Clus-HMC and Clus-HMC-Ens outperform the state-of-the-art methods for gene function prediction on three model organisms:

- CLUS-HMC **outperforms** an existing decision tree learner (C4.5H/M) **w.r.t. predictive performance**, while preserving the interpretability.
- CLUS-HMC-ENS is **more efficient** than statistical learners based on SVMs, while it obtains **competitive results in terms of predictive performance** and is easier to use.

In conclusion, CLUS-HMC and CLUS-HMC-ENS are state-of-the-art tools for gene function prediction. Although decision tree learning is one of the oldest machine learning techniques, decision trees are still competitive in terms of predictive performance and efficiency compared to, for example, SVMs. Moreover, decision tree learning has other advantages such as intuitivity and the fact that decision trees can be interpreted by a domain expert.

## Part II: Structured Input Learning for the Prediction of Molecular Activity

In the second part, we have considered graph mining approaches, which represent learning examples as graphs. As application we have concentrated on predicting the activity of molecules based on the structural arrangement of their atoms and bonds, which is known as learning structure-activity relationships (SAR). Graphs are a popular representation for the atom-bond structure of molecules, and this is the direction we have followed here.

However, since a lot of operations on graphs are NP-hard, many graph mining algorithms are confronted with efficiency issues. In order to improve the efficiency, we exploit specific properties of molecular graphs and change the semantics of the subgraph isomorphism. On the one hand, we use outerplanar graphs to represent molecules. These are graphs that can be drawn in the plane with every vertex adjacent to the outer face. Outerplanar graphs are a suitable class of graphs for molecular datasets, as they are able to represent the majority of molecules. On the other hand, we use the block-and-bridge-preserving (BBP) subgraph isomorphism, which only maps chemically similar parts of the molecules to each other. By restricting the data to outerplanar graphs and employing the BBP subgraph isomorphism, we can achieve a polynomial complexity for the matching of molecular graphs.

First, we have introduced an algorithm that computes a maximum common subgraph (MCS) of two outerplanar graphs under the BBP subgraph isomorphism. We have proved its correctness and shown that it **runs in polynomial time**, both theoretically and empirically. Moreover, the algorithm is orders of magnitude faster than a state-of-the-art MCS algorithm that computes MCSs under the general subgraph isomorphism.

Second, we have constructed an efficiently computable metric that is based on the MCS computed under the BBP subgraph isomorphism. We have investigated

the predictive performance of this metric and of the BBP matching operator in general, showing that:

- Our metric **outperforms** the same metric based on the MCS computed under the general subgraph isomorphism **in terms of predictive performance**. The metric also outperforms a state-of-the-art metric based on molecular fingerprints. We argue that a metric based on the MCS is more intuitive as it uses the original graph structure to compute a distance between molecules.
- At least for molecular datasets, the BBP matching operator can, next to the gain in efficiency, also **improve the predictive performance** of graph mining techniques in general. One reason may be that dealing differently with cycles and linear fragments makes more sense in chemical applications.

Third, we have introduced a simple, direct and efficient approach towards generating features for graphs. It is based on the idea of computing the maximum common subgraph of randomly selected pairs of graphs. We have shown that:

- The approach is **efficient** as it runs in polynomial time.
- It **does not require specifying any parameter**, since one can simply extract all possible pairwise MCSs. Moreover, if a particular number of features is desired, the randomisation approach can be used.
- It yields **better sets of features** than state-of-the-art feature generation methods **in terms of predictive performance, redundancy and feature set size**.

In conclusion, we have proposed efficient graph mining techniques for the classification of molecules. Apart from their efficiency, these techniques also achieve a state-of-the-art predictive performance for SAR learning tasks due to the use of the BBP subgraph isomorphism, which seems to be chemically more relevant than the general subgraph isomorphism.

## 8.2 Further work

In this section, we will formulate some suggestions for further work w.r.t. the two parts of this thesis.

### Part I

From a machine learning viewpoint, there are still several issues related to the HMC task that can be further investigated. For example, an interesting topic is the study of the various evaluation measures for HMC. In this thesis, we have

introduced measures that take an average over all classes of the hierarchy, some taking into account the frequency of the classes and one that does not. The reason for this is that biologists might only be interested in specific functions from the low levels of the hierarchy. Then, the performance on those functions is just as important as the performance on higher level functions. In other situations, however, one wants to obtain good results for all functions in general. Although there is a correlation between the level and the frequency of the function, that is, more specific function tend to have a lower frequency, it is not exactly the same. Therefore, it could be useful to have an evaluation measure that directly takes into account the level of the function. To this end, the hierarchical loss function proposed by Cesa-Bianchi et al. [2006] could be used, but some adaptations are necessary. First, the definition of hierarchical loss only works for trees. For DAGs, a generalisation is needed, as discussed in Sect. 3.5.1. Second, it should be independent from a particular classification threshold. A possible solution here could be to construct a "hierarchical loss curve" that represents the hierarchical loss for selected thresholds between 0 and 1.

Another interesting machine learning task is trying to predict the structure of the hierarchy. Until now, we have always assumed that the hierarchy was fixed. For the functional genomics context, this makes sense, as ontologies are manually constructed by biologists. There may exist, however, other applications of HMC in which this is not the case. To this end, a method could be used to investigate the dependencies of the different classes, possibly with some constraints, for example a maximum number of edges. Then, the quality of the predicted hierarchy could be assessed through the evaluation of the performance of the HMC learner. An alternative would be to exhaustively enumerate all possible structures for the hierarchy and learn a model that predicts one of these structures. However, it is hard to obtain such training data and, given large hierarchies, this approach quickly becomes unfeasible.

In the gene function prediction context, there are also a few important research questions that remain open. For example, it could be investigated what the effect is of combining data from several sources. A relationship may exist between a particular source of data (e.g., microarrays) and a set of functions that are best predicted by it. This could help biologists in order to select the data that should be acquired in order to predict the functions they are interested in.

The effective representation of network information seems to be a key component of successful models for gene function prediction. For the data on mouse, we have performed a very naive approach towards propositionalising network information, but more sophisticated methods could be applied, directly using the information in a protein-protein interaction or metabolics network into the learning method. Some methods have tried to do this, and seem to obtain state-of-the-art results for gene function prediction.

Since it has become easier to determine the genome of higher organisms, which

have a number of genes that is an order of magnitude larger than biology's model organisms, the availability of efficient methods for gene function prediction will become even more important. A technique that may provide a partial solution to this problem is feature selection. Feature selection makes the set of features smaller by discarding uninformative attributes. This can make learning methods more efficient, while it can also improve predictive performance and interpretability by domain experts. Ensembles can be used to carry out some kind of feature selection and select features that are relevant for all classes of a given hierarchy [Breiman, 2001].

## Part II

Also w.r.t. the second part, there are still interesting directions for future research. First of all, various extensions of the feature generation approach are possible. For example, until now, we have only generated features from examples in the training set. For the approaches that do not use information about the labels, also features could be extracted from the test set. This setting is known as semi-supervised learning and it could be interesting to investigate whether this would improve the predictive performance. Instead of the random and class-related example selection strategies, also alternative strategies can be considered, imposing constraints such as the size of examples, or a combination of the existing strategies.

Second, we have only considered outerplanar graphs as a possible class to represent molecules, but there are other classes that may be suitable to represent molecules and for which efficient matching algorithms exist. For example, graphs of bounded treewidth are such a class, and efficient mining algorithms for them have already been presented [Horváth and Ramon, 2008].

Third, we have only focused on methods that use the 2D structure of the molecules. It is known that the performance can be increased by using 3D information as well. However, this is a whole area of research on its own. Finding good representations for 3D structures is one of the important research questions here. Graphs can be used to this end as well. For example, the edges that connect atoms in 3D space can be annotated with distances. However, this results in very dense graphs, for which it is difficult to apply efficient matching algorithms [Kuramochi and Karypis, 2004a; Nowozin and Tsuda, 2008].

Fourth, we have only considered the classification of small molecules, consisting of typically 25 vertices. The next step is to consider proteins, which are an order of magnitude larger. To the best of our knowledge, a few kernel methods have been proposed that can handle proteins efficiently [Shervashidze and Borgwardt, 2009], but at this point, proteins prove too much of a challenge for most mining methods using structural information.

# References


Agrawal, R., T. Imielinski, and A. Swami (1993, May). Mining association rules between sets of items in large databases. In P. Buneman and S. Jajodia (Eds.), *Proceedings of ACM SIGMOD Conference on Management of Data*, Washington, D.C., USA, pp. 207–216. ACM.

Akutsu, T. (1993). A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science E76-A*, 1488–1493.

Altschul, S., T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman (1997). Gapped BLAST and PSI-BLAST: A new generation of protein database search programs. *Nucl. Acids. Res 25*, 3389–3402.

Ashburner, M., C. Ball, J. Blake, D. Botstein, H. Butler, J. Cherry, A. Davis, K. Dolinski, S. Dwight, J. Eppig, M. Harris, D. Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. Matese, J. Richardson, M. Ringwald, G. Rubin, and G. Sherlock (2000). Gene Ontology: Tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics 25*(1), 25–29.

Barutcuoglu, Z., R. Schapire, and O. Troyanskaya (2006). Hierarchical multi-label prediction of gene function. *Bioinformatics 22*(7), 830–836.

Benigni, R. and A. Giuliani (2003). Putting the predictive toxicology challenge into perspective: reflections on the results. *Bioinformatics 19*(10), 1194–1200.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Blockeel, H. (2007). Machine learning and inductive inference: course notes.

Blockeel, H., M. Bruynooghe, S. Džeroski, J. Ramon, and J. Struyf (2002). Hierarchical multi-classification. In *Proceedings of the ACM SIGKDD 2002 Workshop on Multi-Relational Data Mining (MRDM 2002)*, pp. 21–35.

Blockeel, H. and L. De Raedt (1998, June). Top-down induction of first order logical decision trees. *Artificial Intelligence 101*(1-2), 285–297.

Blockeel, H., L. De Raedt, and J. Ramon (1998). Top-down induction of clustering trees. In *Proceedings of the 15th International Conference on Machine Learning*, pp. 55–63.

Blockeel, H., S. Džeroski, and J. Grbović (1999). Simultaneous prediction of multiple chemical parameters of river water quality with Tilde. In *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, Volume 1704 of *Lecture Notes in Artificial Intelligence*, pp. 32–40. Springer.

Blockeel, H., L. Schietgat, J. Struyf, S. Džeroski, and A. Clare (2006). Decision trees for hierarchical multilabel classification: A case study in functional genomics. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, Volume 4213 of *Lecture Notes in Artificial Intelligence*, pp. 18–29.

Bratko, I. (2001). *Prolog Programming for Artificial Intelligence*. Addison-Wesley. 3rd Edition.

Breiman, L. (1996a). Bagging predictors. *Machine Learning 24*(2), 123–140.

Breiman, L. (1996b). Out-of-bag estimation. ftp.stat.berkeley.edu/pub/users/breiman/OOBestimation.ps.

Breiman, L. (2001). Random forests. *Machine Learning 45*(1), 5–32.

Breiman, L., J. Friedman, R. Olshen, and C. Stone (1984). *Classification and Regression Trees*. Belmont: Wadsworth.

Bringmann, B., A. Zimmermann, L. D. Raedt, and S. Nijssen (2006). Don't be afraid of simpler patterns. In *Proceedings of the 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pp. 55–66.

Bunke, H. and K. Shearer (1998). A graph distance metric based on the maximal common subgraph.

Cao, Y., T. Jiang, and T. Girke (2008). A maximum common substructure-based algorithm for searching and predicting drug-like compounds. *Bioinformatics 24*(13), i366–i374.

Caruana, R. (1997). Multitask learning. *Machine Learning 28*, 41–75.

Ceroni, A., F. Costa, and P. Frasconi (2007). Classification of small molecules by two- and three-dimensional decomposition kernels. *Bioinformatics 23*(16), 2038–2045.

Cesa-Bianchi, N., C. Gentile, and L. Zaniboni (2006). Incremental algorithms for hierarchical classification. *Journal of Machine Learning Research 7*, 31–54.

Chaoji, V., M. Al Hasan, S. Salem, J. Besson, and M. J. Zaki (2008). Origami: A novel and effective approach for mining representative orthogonal graph patterns. *Stat. Anal. Data Min. 1*(2), 67–84.

Chen, Y. and D. Xu (2004). Global protein function annotation through mining genome-scale data in yeast saccharomyces cerevisiae. *Nucleic Acids Research 32*(21), 6414–6424.

Chi, Y., R. R. Muntz, S. Nijssen, and J. N. Kok (2005). Frequent subtree mining - an overview. *Fundam. Inform. 66*(1-2), 161–198.

Chu, S., J. DeRisi, M. Eisen, J. Mulholland, D. Botstein, P. Brown, and I. Herskowitz (1998, Oct). The transcriptional program of sporulation in budding yeast. *Science 282*, 699–705.

Chua, H., W. Sung, and L. Wong (2006). Exploiting indirect neighbours and topological weight to predict protein function from protein-protein interactions. *Bioinformatics 22*(13), 1623–1630.

Clare, A. (2003). *Machine learning and data mining for yeast functional genomics*. Ph. D. thesis, University of Wales, Aberystwyth.

Clare, A., A. Karwath, H. Ougham, and R. D. King (2006). Functional bioinformatics for Arabidopsis thaliana. *Bioinformatics 22*(9), 1130–1136.

Clare, A. and R. King (2001). Knowledge discovery in multi-label phenotype data. In L. De Raedt and A. Siebes (Eds.), *5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2001)*, Volume 2168 of *Lecture Notes in Artificial Intelligence*, pp. 42–53. Springer-Verlag.

Clare, A. and R. D. King (2003). Predicting gene function in Saccharomyces cerevisiae. *Bioinformatics 19*(Suppl. 2), ii42–49.

Conte, D., P. Foggia, C. Sansone, and M. Vento (2004). Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence 18*(3), 265–298.

Davis, J. and M. Goadrich (2006). The relationship between precision-recall and ROC curves. In *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233–240.

De Raedt, L. (1998). Attribute-value learning versus inductive logic programming: the missing links (extended abstract). In D. Page (Ed.), *Proceedings of the Eighth International Conference on Inductive Logic Programming*, Volume 1446 of *Lecture Notes in Artificial Intelligence*, pp. 1–8. Springer-Verlag.

De Raedt, L. (2008). *Logical and Relational Learning*. Springer.

De Raedt, L., A. Kimmig, and H. Toivonen (2007). ProbLog: A probabilistic Prolog and its application in link discovery. In M. Veloso (Ed.), *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2462–2467.

De Raedt, L. and J. Ramon (2009). Deriving distance metrics from generality relations. *Pattern Recognition Letters 30*(3), 187–191.

Dehaspe, L. and H. Toivonen (1999). Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery 3*(1), 7–36.

Demšar, D., S. Džeroski, T. Larsen, J. Struyf, J. Axelsen, M. Bruus Pedersen, and P. Henning Krogh (2006). Using multi-objective classification to model communities of soil microarthropods. *Ecological Modelling 191*(1), 131–143.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research 7*(Jan), 1–30.

Deng, M., K. Zhang, S. Mehta, T. Chen, and F. Sun (2002). Prediction of protein function using protein-protein interaction data. In *Proceedings of the IEEE Computer Society Bioinformatics Conference*, pp. 197–206. IEEE Computer Society.

DeRisi, J., V. Iyer, and P. Brown (1997, October). Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science 278*, 680–686.

Deshpande, M., M. Kuramochi, N. Wale, and G. Karypis (2005). Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering 17*(8), 1036–1050.

Diestel, R. (2000). *Graph Theory*. Springer-Verlag.

Dietterich, T. G., R. H. Lathrop, and T. Lozano-Pérez (1997). Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence 89*(1-2), 31–71.

Drucker, H. (1997). Improving regressors using boosting techniques. In *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 107–115.

Džeroski, S. and N. Lavrač (2001a). An introduction to inductive logic programming. In S. Džeroski and N. Lavrač (Eds.), *Relational Data Mining*, pp. 48–74. Springer-Verlag.

Džeroski, S. and N. Lavrač (Eds.) (2001b). *Relational Data Mining*. Springer-Verlag.

Džeroski, S., I. Slavkov, V. Gjorgjioski, and J. Struyf (2006). Analysis of time series data with predictive clustering trees. In *Proceedings of the 5th International Workshop on Knowledge Discovery in Inductive Databases*, pp. 47–58.

Eisen, M., P. Spellman, P. Brown, and D. Botstein (1998, Dec). Cluster analysis and display of genome-wide expression patterns. *Proc. Nat. Acad. Sci. USA 95*, 14863–14868.

Elmasri, R. and S. Navathe (2004). *Fundamentals of Database Systems* (4th ed.). Addison-Wesley.

Expasy (2008). ProtParam. `http://www.expasy.org/tools/protparam.html`.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters 27*, 861–874.

Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth (1996). From data mining to knowledge discovery: An overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 495–515. The MIT Press.

Flach, P. A. and N. Lavrac (2000, July). The role of feature construction in inductive rule learning. In L. D. Raedt and S. Kramer (Eds.), *Proceedings of the ICML2000 workshop on Attribute-Value and Relational Learning: crossing the boundaries*, pp. 1–11. 17th International Conference on Machine Learning.

Garey, M. R. and D. Johnson (1979). *Computers and Intractability: a Guide to the theory of NP-Completeness*. Freeman and Co.

Gärtner, T. (2005). *Kernels for Structured Data*. Ph. D. thesis, University of Bonn, Germany.

Gasch, A., M. Huang, S. Metzner, D. Botstein, S. Elledge, and P. Brown (2001). Genomic expression responses to DNA-damaging agents and the regulatory role of the yeast ATR homolog Mec1p. *Mol. Biol. Cell 12*(10), 2987–3000.

Gasch, A., P. Spellman, C. Kao, O. Carmel-Harel, M. Eisen, G. Storz, D. Botstein, and P. Brown (2000, Dec). Genomic expression program in the response of yeast cells to environmental changes. *Mol. Biol. Cell. 11*, 4241–4257.

Geurts, P., L. Wehenkel, and F. d'Alché Buc (2006). Kernelizing the output of tree-based methods. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, New York, NY, USA, pp. 345–352. ACM.

Gough, J., K. Karplus, R. Hughey, and C. Chothia (2001). Assignment of homology to genome sequences using a library of hidden markov models that represent all proteins of known structure. *Molecular Biology 313*(4), 903–919.

Guan, Y., C. Myers, D. Hess, Z. Barutcuoglu, A. Caudy, and O. Troyanskaya (2008). Predicting gene function in a hierarchical context with an ensemble of classifiers. *Genome Biology 9*(Suppl 1), S3.

Hand, D. J. (2009). Measuring classifier performance: a coherent alternative to the area under the ROC curve. *Machine Learning 77*(1), 103–123.

Hansch, C., P. Maolney, T. Fujita, and M. R.M. (1962). Correlation of biological activity of phenoxyacetic acids with hammett substituent constants and partition coefficients. *Nature 194*, 178–180.

Hayete, B. and J. Bienkowska (2005). GOTrees: Predicting GO associations from protein domain composition using decision trees. In R. B. Altman, T. A. Jung, T. E. Klein, A. K. Dunker, and L. Hunter (Eds.), *Pacific Symposium on Biocomputing*, pp. 127–138. World Scientific.

He, H. and A. K. Singh (2006). Graphrank: Statistical modeling and mining of significant subgraphs in the feature space. In *ICDM '06: Proceedings of the Sixth International Conference on Data Mining*, Washington, DC, USA, pp. 885–890. IEEE Computer Society.

Helma, C., K. S., and D. R. L (2004). Data mining and machine learning techniques for the identification of mutagenicity inducing substructures and structure activity relationships of noncongeneric compounds. *Journal of Chemical Information and Computer Systems 44*(4), 1402–141.

Horváth, T., T. Gärtner, and S. Wrobel (2004). Cyclic pattern kernels for predictive graph mining. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 158–167.

Horváth, T. and J. Ramon (2008, September). Efficient frequent connected subgraph mining in graphs of bounded treewidth. In *Proceedings of Principles and Practice of Knowledge Discovery in Databases 2008*, Volume 5211, Antwerp, Belgium, pp. 520–535. Springer-Verlag.

Horváth, T., J. Ramon, and S. Wrobel (2006, August). Frequent subgraph mining in outerplanar graphs. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Philadelphia, PA, pp. 197–206.

Hughes, T. and F. Roth (2008). A race through the maze of genomic evidence. *Genome Biology 9*(Suppl 1), S1.

Joachims, T. (1999). Making large-scale SVM learning practical. In B. Scholkopf, C. Burges, and A. Smola (Eds.), *Advances in Kernel Methods - Support Vector Learning*. MIT-Press.

Joachims, T. (2002). *Learning to Classify Text using Support Vector Machines: Methods, Theory, and Algorithms*. Springer.

Johnson, M. and G. Maggiora (1990). *Concepts and Applications of Molecular Similarity*. John Wiley.

Karaoz, U., T. Murali, S. Letovsky, Y. Zheng, C. Ding, C. Cantor, and S. Kasif (2004). Whole-genome annotation by using evidence integration in functional-linkage networks. *Proceedings of the National Academy of Sciences 101*(9), 2888–2893.

Karunaratne, T. and H. Boström (2006). Learning to classify structured data by graph propositionalization. In *Proceedings of the Second IASTED International Conference on Computational Intelligence*, pp. 393–398.

Kazius, J., R. McGuire, and R. Bursi (2005). Derivation and validation of toxicophores for mutagenicity prediction. *Journal of Medicinal Chemistry 48*(1), 312–320.

Kim, W., C. Krumpelman, and E. Marcotte (2008). Inferring mouse gene functions from genomic-scale data using a combined functional network/classification strategy. *Genome Biology 9*(Suppl 1), S5.

King, R., S. Muggleton, A. Srinivasan, and M. Sternberg (1996). Structure-activity relationships derived by machine learning: The use of atoms and their bond connectivities to predict mutagenicity by inductive logic programming. *Proceedings of the National Academy of Sciences 93*, 438–442.

King, R. D. (2004). Applying inductive logic programming to predicting gene function. *AI Mag. 25*(1), 57–68.

King, R. D., A. Srinivasan, and L. Dehaspe (2001, feb). Warmr: a data mining tool for chemical data. *Journal of Computer-Aided Molecular Design 15*(2), 173–181.

Kocev, D., C. Vens, J. Struyf, and S. Džeroski (2007). Ensembles of multi-objective decision trees. In *Proceedings of the 18th European Conference on Machine Learning*, pp. 624–631. Springer.

Koch, I. (2001). Enumerating all connected maximal common subgraphs in two graphs. *Theor. Comput. Sci. 250*(1-2), 1–30.

Koller, D. and M. Sahami (1997). Hierarchically classifying documents using very few words. In *Proceedings of ICML-97, 14th International Conference on Machine Learning*, Nashville, TN, USA, pp. 170–178. Morgan Kaufmann.

Kramer, S., L. De Raedt, and C. Helma (2001). Molecular feature mining in HIV data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-01)*, pp. 136–143. ACM Press.

Kramer, S., N. Lavrač, and P. Flach (2001). Propositionalization approaches to relational data mining. In S. Džeroski and N. Lavrač (Eds.), *Relational Data Mining*, pp. 262–291. Springer-Verlag.

Kramer, S. and G. Widmer (2001). Inducing classification and regression trees in first order logic. In S. Džeroski and N. Lavrač (Eds.), *Relational Data Mining*, pp. 140–159. Springer-Verlag.

Kumar, A., K. Cheung, P. Ross-Macdonald, P. Coelho, P. Miller, and M. Snyder (2000). TRIPLES: A database of gene function in S. cerevisiae. *Nucleic Acids Res. 28*, 81–84.

Kuramochi, M. and G. Karypis (2004a). Discovering frequent geometric subgraphs. In *Proceedings of the 2004 IEEE International Conference on Data Mining*, pp. 258–265.

Kuramochi, M. and G. Karypis (2004b). An efficient algorithm for discovering frequent subgraphs. *IEEE Trans. Knowl. Data Eng. 16*(9), 1038–1051.

Lanckriet, G. R., M. Deng, N. Cristianini, M. I. Jordan, and W. S. Noble (2004). Kernel-based data fusion and its application to protein function prediction in yeast. In *Proceedings of the Pacific Symposium on Biocomputing*, pp. 300–311.

Lavrač, N., S. Džeroski, and M. Grobelnik (1991). Learning nonrecursive definitions of relations with LINUS. In Y. Kodratoff (Ed.), *Proceedings of the Fifth European Working Session on Learning*, Volume 482 of *Lecture Notes in Artificial Intelligence*, pp. 265–281. Springer-Verlag.

Lee, H., Z. Tu, M. Deng, F. Sun, and T. Chen (2006). Diffusion kernel-based logistic regression models for protein function prediction. *OMICS 10*(1), 40–55.

Li, Y., D. Blostein, and P. Abolmaesumi (2005). *Graph-Based Representations in Pattern Recognition*, Chapter Asymmetric Inexact Matching of Spatially-Attributed Graphs, pp. 253–262. Springer.

Lingas, A. (1989). Subgraph isomorphism for biconnected outerplanar graphs in cubic time. *Theoretical Computer Science 63*, 295–302.

Maunz, A., C. Helma, and S. Kramer (2009). Large-scale graph mining using backbone refinement classes. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, USA, pp. 617–626. ACM.

McCarthy, J., M. Minsky, N. Rochester, and C. Shannon (1955). A proposal for the dartmouth summer research project on artificial intelligence.

McGregor, J. J. (1982). Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience 12*, 23–34.

Menchetti, S., F. Costa, and P. Frasconi (2005). Weighted decomposition kernels. In *Proceedings of the 22nd International Conference on Machine Learning*, pp. 585–592.

Mewes, H., K. Heumann, A. Kaps, K. Mayer, F. Pfeiffer, S. Stocker, and D. Frishman (1999). MIPS: A database for protein sequences and complete genomes. *Nucleic Acids Research 27*, 44–48.

Mitchell, S. L. (1979). Linear algorithms to recognize outerplanar and maximal outerplanar graphs. *Information Processing Letters 9*(5), 229–232.

Mitchell, T. (1997). *Machine Learning*. McGraw-Hill.

Mostafavi, S., D. Ray, D. Warde-Farley, C. Grouios, and Q. Morris (2008). GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. *Genome Biology 9*(Suppl 1), S4.

Muggleton, S. and L. De Raedt (1994). Inductive logic programming : Theory and methods. *Journal of Logic Programming 19,20*, 629–679.

Munkres, J. (1957). Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics 5*(1), 32–38.

Nijssen, S. and J. N. Kok (2004). A quickstart in frequent structure mining can make a difference. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pp. 647–652.

Nowozin, S. and K. Tsuda (2008). Frequent subgraph retrieval in geometric graph databases. In *Proceedings of the 2008 IEEE International Conference on Data Mining*, pp. 953–958.

Obozinski, G., G. Lanckriet, C. Grant, M. Jordan, and W. Noble (2008). Consistent probabilistic outputs for protein function prediction. *Genome Biology 9*(Suppl 1), S6.

Oliver, S. (1996). A network approach to the systematic analysis of yeast gene function. *Trends in Genetics 12(7)*, 241–242.

Ouali, M. and R. King (2000, Jun). Cascaded multiple classifiers for secondary structure prediction. *Protein Science 9(6)*, 1162–76.

Page, D. and M. Craven (2003). Biological applications of multi-relational data mining. *SIGKDDEX 5*(1), 69–79.

Pena-Castillo, L., M. Tasan, C. Myers, H. Lee, T. Joshi, C. Zhang, Y. Guan, M. Leone, P. A., W. Kim, C. Krumpelman, W. Tian, G. Obozinski, Y. Qi, S. Mostafavi, G. Lin, G. Berriz, F. Gibbons, G. Lanckriet, J. Qiu, C. Grant, Z. Barutcuoglu, D. Hill, D. Warde-Farley, C. Grouios, D. Ray, J. Blake, M. Deng, M. Jordan, W. Noble, Q. Morris, J. Klein-Seetharaman, Z. Bar-Joseph, T. Chen, F. Sun, O. Troyanskaya, E. Marcotte, D. Xu, T. Hughes, and F. Roth (2008). A critical assessment of Mus musculus gene function prediction using integrated genomic evidence. *Genome Biology 9*(Suppl 1), S2.

Provost, F. and T. Fawcett (1998). Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pp. 43–48. AAAI Press.

Provost, F. J. and T. Fawcett (2001). Robust classification for imprecise environments. *Machine Learning 42*(3), 203–231.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann series in Machine Learning. Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning 1*, 81–106.

Raymond, J., E. Gardiner, and P. Willett (2002). Rascal: Calculation of graph similarity using maximum common edge subgraphs. *The Computer Journal 45*, 631–644.

Raymond, J. and P. Willett (2002a). Effectiveness of graph-based and fingerprint-based similarity measures for virtual screening of 2D chemical structure databases. *Journal of Computer-Aided Design 16*, 59–71.

Raymond, J. and P. Willett (2002b). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design 16*, 521–533.

Roth, F., J. Hughes, P. Estep, and G. Church (1998, October). Fining DNA regulatory motifs within unaligned noncoding sequences clustered by whole-genome mRNA quantitation. *Nature Biotechnology 16*, 939–945.

Rousu, J., C. Saunders, S. Szedmak, and J. Shawe-Taylor (2006). Kernel-based learning of hierarchical multilabel classification models. *Journal of Machine Learning Research 7*, 1601–1626.

Schietgat, L., F. Costa, J. Ramon, and L. De Raedt (2010). Effective feature construction by maximum common subgraph sampling. *Machine Learning*. submitted.

Schietgat, L., J. Ramon, and M. Bruynooghe (2007). A polynomial-time metric for outerplanar graphs. In *Benelearn 2007: Proceedings of the Annual Machine Learning Conference of Belgium and the Netherlands*, pp. 97–104.

Schietgat, L., J. Ramon, M. Bruynooghe, and H. Blockeel (2008). An efficiently computable graph-based metric for the classification of small molecules. In *Proceedings of the 11th International Conference on Discovery Science*, Volume 5255 of *Lecture Notes in Artificial Intelligence*, pp. 197–209.

Schietgat, L., C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Džeroski (2010). Predicting gene function using hierarchical multi-label decision tree ensembles. *BMC Bioinformatics 11*(2).

Sebag, M. (1997). Distance induction in first order logic. In N. Lavrač and S. Džeroski (Eds.), *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, Volume 1297 of *Lecture Notes in Artificial Intelligence*, pp. 264–272. Springer.

Shamir, R. and D. Tsur (1992, November). Faster subtree isomorphism. *Journal of Algorithms 33*(2), 267–280.

Shearer, K., H. Bunke, and S. Venkatesh (2001). Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition 34*(5), 1075–1091.

Shervashidze, N. and K. Borgwardt (2009). Fast subtree kernels on graphs. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta (Eds.), *Advances in Neural Information Processing Systems 22*, pp. 1660–1668.

Spellman, P., G. Sherlock, M. Zhang, V. Iyer, K. Anders, M. Eisen, P. Brown, D. Botstein, and B. Futcher (1998, December). Comprehensive identification of cell cycle-regulated genes of the yeast *Saccharomyces cerevisiae* by microarray hybridization. *Molecular Biology of the Cell 9*, 3273–3297.

Srinivasan, A., R. King, S. Muggleton, and M. Sternberg (1997). Carcinogenesis predictions using ILP. In *Proceedings of the Seventh International Workshop on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence, pp. 273–287. Springer-Verlag.

Srinivasan, A. and R. D. King (1999). Feature construction with inductive logic programming: A study of quantitative predictions of biological activity aided by structural attributes. *Data Min. Knowl. Discov. 3*(1), 37–57.

Srinivasan, A., S. Muggleton, M. Sternberg, and R. King (1996). Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence 85*(1,2), 277–299.

Stenger, B., A. Thayananthan, P. Torr, and R. Cipolla (2007). Estimating 3D hand pose using hierarchical multi-label classification. *Image and Vision Computing 5*(12), 1885–1894.

Struyf, J. and S. Džeroski (2006). Constraint based induction of multi-objective regression trees. In *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID'05, Revised, Selected and Invited Papers*, Volume 3933 of *Lecture Notes in Computer Science*, pp. 222–233.

Struyf, J. and S. Džeroski (2007). Clustering trees with instance level constraints. In *Proceedings of the 18th European Conference on Machine Learning*, Volume 4701 of *Lecture Notes in Computer Science*, pp. 359–370. Springer.

Struyf, J., S. Džeroski, H. Blockeel, and A. Clare (2005). Hierarchical multi-classification with predictive clustering trees in functional genomics. In *Progress in Artificial Intelligence: 12th Portugese Conference on Artificial Intelligence*, Volume 3808 of *Lecture Notes in Computer Science*, pp. 272–283. Springer.

Swamidass, S. J., J. Chen, J. Bruand, P. Phung, L. Ralaivola, and P. Baldi (2005). Kernels for small molecules and the prediction of mutagenicity, toxicity and anti-cancer activity. *Bioinformatics 21*(suppl_1), i359–368.

Syslo, M. (1982). The subgraph isomorphism problem for outerplanar graphs. *Theoretical Computer Science 17*(1), 91–97.

Taskar, B., C. Guestrin, and D. Koller (2003). Max-margin Markov networks. In *Advances in Neural Information Processing Systems*, Volume 16. MIT Press.

Tian, W., L. Zhang, M. Tasan, F. Gibbons, O. King, J. Park, Z. Wunderlich, J. Cherry, and F. Roth (2008). Combining guilt-by-association and guilt-by-profiling to predict saccharomyces cerevisiae gene function. *Genome Biology 9*(Suppl 1), S7.

Toivonen, H., A. Srinivasan, R. D. King, S. Kramer, and C. Helma (2003). Statistical evaluation of the predictive toxicology challenge 2000-2001. *Bioinformatics 19*(10), 1183–1193.

Troyanskaya, O., K. Dolinski, A. Owen, R. Altman, and B. D. (2003). A bayesian framework for combining heterogeneous data sources for gene function prediction (in saccharomyces cerevisiae). *Proceedings of the National Academy of Sciences 100*(14), 8348–8353.

Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun (2005). Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res. 6*, 1453–1484.

Tsoumakas, G. and I. Vlahavas (2007). Random k-labelsets: An ensemble method for multilabel classification. In *Proceedings of the 18th European Conference on Machine Learning*, Volume 4701 of *Lecture Notes in Computer Science*, pp. 406–417. Springer.

Van Assche, A. and H. Blockeel (2007). Seeing the forest through the trees: Learning a comprehensible model from an ensemble. In *Proceedings of The 18th European Conference on Machine Learning*, Volume 4701 of *Lecture Notes in Artificial Intelligence*, pp. 418–429. Springer.

Vens, C., J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel (2008). Decision trees for hierarchical multi-label classification. *Machine Learning 73*(2), 185–214.

Wale, N., I. Watson, and G. Karypis (2008). Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowledge and Information Systems 14*, 347–375.

Watanabe, S. (1960). Information theoretical analysis of multivariate correlation. *IBM Journal of Research and Development 4*(1), 66–82.

Watson, J. D. and F. H. C. Crick (1953). Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature 171*, 737–738.

Weiss, G. and F. Provost (2003). Learning when training data are costly: The effect of class distribution on tree induction. *Journal of Artificial Intelligence Research 19*, 315–354.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics 1*, 80–83.

Willett, P. (2006). Similarity-based virtual screening using 2D fingerprints. *Drug Discovery Today 11*(23/24), 1046–1051.

Witten, I. and E. Frank (2005). *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann. 2nd Edition.

Yan, X. and J. Han (2002). gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, Japan, pp. 721–724. IEEE Computer Society.

Yang, Y. (1999). An evaluation of statistical approaches to text categorization. *Information Retrieval 1*, 69–90.

Zantema, H. and H. Bodlaender (2000). Finding small equivalent decision trees is hard. *International Journal of Foundations of Computer Science 11*(2), 343–354.

Zdobnov, E. and R. Apweiler (2001). Interproscan - an integration platform for the signature-recognition methods in interpro. *Bioinformatics 17*(9), 847–848.

# Index

accuracy, 30
attribute, 14
attribute-value data, 14
attribute-value learning, 22

biconnected graph, 103
block, 104
block-and-bridge-preserving subgraph isomorphism, 105
bridge, 104

classification, 22
connected graph, 103
contingency matrix, 30

decision tree, 26
decision tree learning, 25
descriptive learning, 5, 22
directed acyclic graph, 17

graph, 15
graph isomorphism, 18, 104
graph mining, 24
graph size, 103

inductive logic programming, 23

labeled graph, 16

maximum common subgraph, 106
metric, 133
model, 22

nominal attribute, 14
numerical attribute, 14

ordinal attribute, 14
outerplanar graph, 104

pattern mining, 24
planar graph, 103
precision, 31
predictive learning, 5, 22
propositionalisation, 24
pseudo-metric, 133

ranking, 22
recall, 31
regression, 22
relational learning, 23

sequence, 103
single network setting, 24
structure-activity relationship learning, 99
structured data, 13
subgraph isomorphism, 18, 105

top-down induction of decision trees, 28
transactional setting, 24

weighted maximal matching, 106

# Publication List

## Articles in internationally reviewed scientific journals

- L. Schietgat, F. Costa, J. Ramon, and L. De Raedt. Effective feature construction by maximum common subgraph sampling. *Machine Learning*, to appear. Springer. 2010.

- L. Schietgat, C. Vens, J. Struyf, H. Blockeel, D. Kocev, and S. Džeroski. Predicting gene function using hierarchical multi-label decision tree ensembles. *BMC Bioinformatics*, vol. 11 (2). BioMed Central. 2010.
`https://lirias.kuleuven.be/handle/123456789/261164`

- C. Vens, J. Struyf, L. Schietgat, S. Džeroski, and H. Blockeel. Decision trees for hierarchical multi-label classification. *Machine Learning*, vol. 73 (2), pp. 185–214. Springer. 2008.
`https://lirias.kuleuven.be/handle/123456789/186698`

## Contributions at international conferences and workshops

### Published in *Lecture Notes in Computer Science*

- L. Schietgat, J. Ramon, M. Bruynooghe, and H. Blockeel. An efficiently computable graph-based metric for the classification of small molecules. In *Proceedings of the 11th International Conference on Discovery Science (DS)*, vol. 5255 of *Lecture Notes in Computer Science*, pp. 197–209, Springer. 2008.
`https://lirias.kuleuven.be/handle/123456789/186720`

- H. Blockeel, L. Schietgat, J. Struyf, S. Džeroski, and A. Clare. Decision trees for hierarchical multi-label classification: A case study in functional genomics. In *Proceedings of the 10th European Conference on Principles*

*and Practice in Knowledge Discovery (PKDD)*, vol. 4213 of *Lecture Notes in Computer Science*, pp. 18–29, Springer. 2006.
https://lirias.kuleuven.be/handle/123456789/124946

## Published in conference proceedings

- L. Schietgat, F. Costa, J. Ramon, and L. De Raedt. Maximum common subgraph mining: A fast and effective approach towards feature generation. In *Proceedings of the 7th International Workshop on Mining and Learning with Graphs (MLG)*, pp. 1–3. 2009.
https://lirias.kuleuven.be/handle/123456789/241347

- L. Schietgat, K. Theys, J. Ramon, H. Blockeel, and A. Vandamme. Distinguishing epidemiological dependent from treatment (resistance) dependent HIV mutations: problem statement. In *Proceedings of the 1st International Workshop on Statistical and Relational Learning in Bioinformatics (ECML-PKDD)*, pp. 1–6. 2008.
https://lirias.kuleuven.be/handle/123456789/197472

- L. Schietgat, J. Ramon, and M. Bruynooghe. A polynomial-time metric for outerplanar graphs. In *Proceedings of the 5th International Workshop on Mining and Learning with Graphs (MLG)*, pp. 67–70. 2007.
https://lirias.kuleuven.be/handle/123456789/176101

- L. Schietgat, J. Ramon, and M. Bruynooghe. A polynomial-time metric for outerplanar graphs. In *Proceedings of the 16th Machine Learning Conference of Belgium and the Netherlands (Benelearn)*, pp. 97–104. 2007.
https://lirias.kuleuven.be/handle/123456789/146129

- J. Ramon, T. Horvath, L. Schietgat, and S. Wrobel. FOG: Finding Outerplanar Graphs. Demo session of the ACM SIGKDD Conference (KDD), pp. 1–3. 2006.
https://lirias.kuleuven.be/handle/123456789/133369

- H. Blockeel, L. Schietgat, J. Struyf, A. Clare, S. Džeroski. Hierarchical multi-label classification trees for gene function prediction (Extended abstract). In *Proceedings of the International Workshop on Probabilistic Modeling and Machine Learning in Structural and Systems Biology (PMSB 2006)*, pp. 9-14. 2006.
https://lirias.kuleuven.be/handle/123456789/134154

# Biography

Leander Schietgat was born on July 21, 1983 in Kortrijk, Belgium. After finishing high school at the Sint-Jozefinstituut Kortrijk in 2001, he started studying informatics (licentiaat informatica) at the Katholieke Universiteit Leuven, Campus Kortrijk. After two years, he came to Leuven to graduate magna cum laude as Master in Informatics in 2005. His master thesis 'Relational data mining of DNA sequences' was supervised by Hendrik Blockeel.

In August 2005, he joined the Declarative Languages and Artificial Intelligence group of the Department of Computer Science of the K.U.Leuven. Since January 2006, his research was funded by a personal grant of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT Vlaanderen). His Ph.D. research, entitled 'Graph-based data mining for biological applications', was supervised by Hendrik Blockeel and Maurice Bruynooghe and was defended on May 28, 2010.